<table>
<thead>
<tr>
<th>1 Entry points</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 Java entry points</td>
<td>5</td>
</tr>
<tr>
<td>3 Public API</td>
<td>7</td>
</tr>
<tr>
<td>4 All packages</td>
<td>9</td>
</tr>
<tr>
<td>4.1 robot Package</td>
<td>9</td>
</tr>
<tr>
<td>4.2 api Package</td>
<td>235</td>
</tr>
<tr>
<td>4.3 common Package</td>
<td>236</td>
</tr>
<tr>
<td>4.4 conf Package</td>
<td>240</td>
</tr>
<tr>
<td>4.5 htmldata Package</td>
<td>241</td>
</tr>
<tr>
<td>4.6 libdocpkg Package</td>
<td>243</td>
</tr>
<tr>
<td>4.7 libraries Package</td>
<td>245</td>
</tr>
<tr>
<td>4.8 model Package</td>
<td>372</td>
</tr>
<tr>
<td>4.9 output Package</td>
<td>382</td>
</tr>
<tr>
<td>4.10 parsing Package</td>
<td>390</td>
</tr>
<tr>
<td>4.11 reporting Package</td>
<td>403</td>
</tr>
<tr>
<td>4.12 result Package</td>
<td>410</td>
</tr>
<tr>
<td>4.13 running Package</td>
<td>424</td>
</tr>
<tr>
<td>4.14 utils Package</td>
<td>434</td>
</tr>
<tr>
<td>4.15 variables Package</td>
<td>449</td>
</tr>
<tr>
<td>4.16 writer Package</td>
<td>451</td>
</tr>
<tr>
<td>5 Indices</td>
<td>455</td>
</tr>
<tr>
<td>Python Module Index</td>
<td>457</td>
</tr>
</tbody>
</table>
This document describes the public API of Robot Framework. Installation, basic usage and wealth of other topics are covered in Robot Framework User Guide.

Main API entry points are documented here, but the lower level implementation details are not that well documented. If the documentation is insufficient, it is possible to view the source code by clicking [source] link in the documentation. In case viewing the source is not helpful either, questions may be sent to the users mailing list.
Entry points

Command line entry points are implemented as Python modules and they also provide programmatic APIs. Following entry points exist:

- `robot.run` entry point for executing tests.
- `robot.rebot` entry point for post-processing outputs.
- `robot.libdoc` entry point for the `libdoc` tool.
- `robot.testdoc` entry point for the `testdoc` tool.
- `robot.tidy` entry point for the `tidy` tool.
Java entry points

The Robot Framework Jar distribution contains also a Java API, in the form of the org.robotframework.RobotFramework class.
This package exposes the public APIs of Robot Framework.

Unless stated otherwise, the APIs exposed in this module are considered stable, and thus safe to use when building external tools on top of Robot Framework.

Currently exposed APIs are:

- `logger` for test libraries' logging purposes.
- `ExecutionResult()` for reading execution results from XML output files.
- `TestCaseFile`, `TestDataDirectory`, and `ResourceFile` for parsing data files. In addition, a convenience factory function `TestData()` creates either `TestCaseFile` or `TestDataDirectory` based on the input.
- `TestSuite()` for creating a test suite that can be executed. This API is going to change in Robot Framework 2.8.

These names can be imported like this:

```python
from robot.api import <name>
```

See documentations of the individual APIs for more details.
All robot packages are listed below. Typically you should not need to import anything from them directly, but the above public APIs may return objects implemented in them.

### 4.1 robot Package

#### 4.1.1 robot Package

`robot.__init__.run (*datasources, **options)`

Executes given Robot Framework data sources with given options.

Data sources are paths to files and directories, similarly as when running pybot/jybot from the command line. Options are given as keywords arguments and their names are same as long command line options without hyphens.

Options that can be given on the command line multiple times can be passed as lists like `include=['tag1', 'tag2']`. Starting from 2.7.2, when such option is used only once, it can be given also as a single string like `include='tag'`.

To capture stdout and/or stderr streams, pass open file objects in as special keyword arguments `stdout` and `stderr`, respectively.

A return code is returned similarly as when running on the command line.

Example:

```python
run('path/to/tests.html', include=['tag1', 'tag2'])
with open('stdout.txt', 'w') as stdout:
    run('t1.txt', 't2.txt', report='r.html', log='NONE', stdout=stdout)
```

Equivalent command line usage:

```bash
pybot --include tag1 --include tag2 path/to/tests.html
pybot --report r.html --log NONE t1.txt t2.txt > stdout.txt
```

`robot.__init__.run_cli (arguments)`

Command line execution entry point for running tests.

For programmatic usage the `run()` method is typically better. It has better API for that usage and does not call `sys.exit()` like this method.

`robot.__init__.rebot (*datasources, **options)`

Creates reports/logs from given Robot output files with given options.
Given input files are paths to Robot output files similarly as when running rebot from the command line. Options are given as keywords arguments and their names are same as long command line options without hyphens.

Options that can be given on the command line multiple times can be passed as lists like `include=['tag1', 'tag2']`. Starting from 2.7.2, when such option is used only once, it can be given also as a single string like `include='tag'`.

To capture stdout and/or stderr streams, pass open file objects in as special keyword arguments `stdout` and `stderr`, respectively.

A return code is returned similarly as when running on the command line.

Examples:

```python
rebot('path/to/output.xml')
with open('stdout.txt', 'w') as stdout:
    rebot('o1.xml', 'o2.xml', report='r.html', log='NONE', stdout=stdout)
```

Equivalent command line usage:

```bash
rebot path/to/output.xml
rebot --report r.html --log NONE o1.xml o2.xml > stdout.txt
```

`robot.__init__.rebot_cli(arguments)`

Command line execution entry point for running rebot.

For programmatic usage the `rebot()` method is typically better. It has better API for that usage and does not call `sys.exit()` like this method.

### 4.1.2 errors Module

`exception` `robot.errors.RobotError(message=''); details='')`

Bases: `exceptions.Exception`

Base class for Robot Framework errors.

Do not raise this method but use more specific errors instead.

- `message`
- `args`

`exception` `robot.errors.FrameworkError(message=''); details='')`

Bases: `robot.errors.RobotError`

Can be used when the core framework goes to unexpected state.

It is good to explicitly raise a FrameworkError if some framework component is used incorrectly. This is pretty much same as ‘Internal Error’ and should of course never happen.

- `args`
- `message`

`exception` `robot.errors.DataError(message=''); details='')`

Bases: `robot.errors.RobotError`

Used when the provided test data is invalid.

DataErrors are not be caught by keywords that run other keywords (e.g. Run Keyword And Expect Error). Libraries should thus use this exception with care.

- `args`
**Exception** robot.errors.TimeoutError

* Bases: robot.errors.RobotError

Used when a test or keyword timeout occurs.

This exception is handled specially so that execution of the current test is always stopped immediately and it is not caught by keywords executing other keywords (e.g. `Run Keyword And Expect Error`). Libraries should thus NOT use this exception themselves.

**Args**
- `message`
- `details`

**Exception** robot.errors.Information

* Bases: robot.errors.RobotError

Used by argument parser with –help or –version.

**Args**
- `message`
- `details`

**Exception** robot.errors.ExecutionFailed

* Bases: robot.errors.RobotError

Used for communicating failures in test execution.

**Methods**
- `message`
- `dont_cont`
- `cont`
- `can_continue` (teardown=False, templated=False, dry_run=False)
- `get_errors()`

**Exception** robot.errors.HandlerExecutionFailed

* Bases: robot.errors.ExecutionFailed

**Methods**
- `cont`
- `args`
- `can_continue` (teardown=False, templated=False, dry_run=False)
- `dont_cont`
- `get_errors()`

**Exception** robot.errors.ExecutionFailures

* Bases: robot.errors.ExecutionFailed

**Methods**
- `message`
- `get_errors()`
- `args`
- `can_continue` (teardown=False, templated=False, dry_run=False)
- `cont`
exception robot.errors.UserKeywordExecutionFailed (run_errors=None, teardown_errors=None)

Bases: robot.errors.ExecutionFailures

message

args
can_continue (teardown=False, templated=False, dry_run=False)

cont
dont_cont

get_errors ()

exception robot.errors.RemoteError (message='', details='')

Bases: robot.errors.RobotError

Used by Remote library to report remote errors.

args

message

4.1.3 jarrunner Module

4.1.4 jythonworkarounds Module

4.1.5 libdoc Module

class robot.libdoc.LibDoc

Bases: robot.utils.application.Application

validate (options, arguments)

main (args, name='', version='', format=None, docformat=None)

console (msg)

execute (*arguments, **options)

execute_cli (cli_arguments)

robot.libdoc.libdoc_cli (args)

Executes libdoc similarly as from the command line.

Parameters args – command line arguments as a list of strings.

Example: libdoc_cli([-name’, ‘Something’, ‘MyLibrary.py’, ‘doc.html’])

robot.libdoc.libdoc (library_or_resource, outfile, name='', version='', format=None)

Executes libdoc.

Arguments are same as command line options to libdoc.py.

Example: libdoc(’MyLibrary.py’, ‘MyLibrary.html’, version=’1.0’)
4.1.6 pythonpathsetter Module

Module that adds directories needed by Robot to sys.path when imported.

```python
robot.pythonpathsetter.add_path(path, end=False)
robot.pythonpathsetter.remove_path(path)
```

4.1.7 rebot Module

class robot.rebot.Rebot
   Bases: robot.run.RobotFramework
   main(datasources, **options)
   console(msg)
   execute(*arguments, **options)
   execute_cli(cli_arguments)
   validate(options, arguments)
robot.rebot.rebot_cli(arguments)

Command line execution entry point for running rebot.

For programmatic usage the rebot() method is typically better. It has better API for that usage and does not call sys.exit() like this method.

```python
robot.rebot(*datasources, **options)
```

Creates reports/logs from given Robot output files with given options.

Given input files are paths to Robot output files similarly as when running rebot from the command line. Options are given as keywords arguments and their names are same as long command line options without hyphens.

Options that can be given on the command line multiple times can be passed as lists like include=['tag1', 'tag2']. Starting from 2.7.2, when such option is used only once, it can be given also as a single string like include='tag'.

To capture stdout and/or stderr streams, pass open file objects in as special keyword arguments `stdout` and `stderr`, respectively.

A return code is returned similarly as when running on the command line.

Examples:

```python
rebot('path/to/output.xml')
with open('stdout.txt', 'w') as stdout:
    rebot('o1.xml', 'o2.xml', report='r.html', log='NONE', stdout=stdout)
```

Equivalent command line usage:

```bash
rebot path/to/output.xml
rebot --report r.html --log NONE o1.xml o2.xml > stdout.txt
```

4.1.8 run Module

class robot.run.RobotFramework
   Bases: robot.utils.application.Application
   main(datasources, **options)
**validate**(options, arguments)

**console**(msg)

**execute**(*arguments, **options*)

**execute_cli**(cli_arguments)

robot.run.**run_cli**(arguments)
  Command line execution entry point for running tests.

  For programmatic usage the **run()** method is typically better. It has better API for that usage and does not call sys.exit() like this method.

robot.run.**run**(*datasources, **options*)
  Executes given Robot Framework data sources with given options.

  Data sources are paths to files and directories, similarly as when running pybot/jybot from the command line. Options are given as keywords arguments and their names are same as long command line options without hyphens.

  Options that can be given on the command line multiple times can be passed as lists like `include=['tag1', 'tag2']`. Starting from 2.7.2, when such option is used only once, it can be given also as a single string like `include='tag'`.

  To capture stdout and/or stderr streams, pass open file objects in as special keyword arguments `stdout` and `stderr`, respectively.

  A return code is returned similarly as when running on the command line.

  Example:

  ```python
  run('path/to/tests.html', include=['tag1', 'tag2'])
  with open('stdout.txt', 'w') as stdout:
    run('t1.txt', 't2.txt', report='r.html', log='NONE', stdout=stdout)
  ```

  Equivalent command line usage:

  ```bash
  pybot --include tag1 --include tag2 path/to/tests.html
  pybot --report r.html --log NONE t1.txt t2.txt > stdout.txt
  ```

### 4.1.9 runner Module

### 4.1.10 testdoc Module

class robot.testdoc.TestDoc
  Bases: robot.utils.application.Application

  **main**(datasources, title=None, **options)

  **console**(msg)

  **execute**(*arguments, **options*)

  **execute_cli**(cli_arguments)

  **validate**(options, arguments)

class robot.testdoc.TestdocModelWriter(output, suite, title=None)
  Bases: robot.htmldata.htmlfilewriter.ModelWriter

  **write**(line)
write_data()

handles(line)

class robot.testdoc.JsonConverter(output_path=None)
    Bases: object
    convert(suite)

robot.testdoc.testdoc_cli(args)
    Executes testdoc similarly as from the command line.
    
    Parameters args – command line arguments as a list of strings.
    
    Example: testdoc_cli(['–title', 'Test Plan', 'mytests', 'plan.html'])

4.1.11 tidy Module

class robot.tidy.Tidy(**options)
    Bases: object
    file(path, output=None)
    directory(path)
    inplace(path)

class robot.tidy.TidyCommandLine
    Bases: robot.utils.application.Application
    main(arguments, recursive=False, inplace=False, format='txt', usepipes=False, spacecount=4, lineseparator='n')
    validate(opts, args)
    console(msg)
    execute(*arguments, **options)
    execute_cli(cli_arguments)

class robot.tidy.ArgumentValidator
    Bases: object
    mode_and_arguments(args, recursive, inplace, **others)
    format(args, format, inplace, recursive, **others)
    line_sep(lineseparator, **others)
    spacecount(spacecount)

robot.tidy.tidy_cli(args)
    Executes tidy similarly as from the command line.
    
    Parameters args – command line arguments as a list of strings.
    
    Example: tidy_cli(['–format', 'txt', 'mytests.html'])
4.1.12 version Module

robot.version.get_version(sep='')
robot.version.get_full_version(who='')

4.1.13 Subpackages

api Package

This package exposes the public APIs of Robot Framework.

Unless stated otherwise, the APIs exposed in this module are considered stable, and thus safe to use when building external tools on top of Robot Framework.

Currently exposed APIs are:

- `logger` for test libraries’ logging purposes.
- `ExecutionResult()` for reading execution results from XML output files.
- `TestCaseFile`, `TestDataDirectory`, and `ResourceFile` for parsing data files. In addition, a convenience factory function `TestData()` creates either `TestCaseFile` or `TestDataDirectory` based on the input.
- `TestSuite()` for creating a test suite that can be executed. This API is going to change in Robot Framework 2.8.

These names can be imported like this:

```python
from robot.api import <name>
```

See documentations of the individual APIs for more details.

**logger Module**

Public logging API for test libraries.

This module provides a public API for writing messages to the log file and the console. Test libraries can use this API like `logger.info('My message')` instead of logging through the standard output like `print '*INFO* My message'`. In addition to a programmatic interface being cleaner to use, this API has a benefit that the log messages have accurate timestamps.

**Log levels** It is possible to log messages using levels `TRACE`, `DEBUG`, `INFO` and `WARN` either using the `write` method or, more commonly, with the log level specific `trace`, `debug`, `info` and `warn` methods.

By default the trace and debug messages are not logged but that can be changed with the `-loglevel` command line option. Warnings are automatically written also to the Test Execution Errors section in the log file and to the console.

**Logging HTML** All methods that are used for writing messages to the log file have an optional `html` argument. If a message to be logged is supposed to be shown as HTML, this argument should be set to `True`. 
Example  from robot.api import logger
def my_keyword(arg):
    logger.debug('Got argument %s' % arg)
do_something()
logger.info('This is a boring example', html=True)


robot.api.logger.write(msg, level, html=False)
    Writes the message to the log file using the given level.
    Valid log levels are TRACE, DEBUG, INFO and WARN. Instead of using this method, it is generally better to use
    the level specific methods such as info and debug.

robot.api.logger.trace(msg, html=False)
    Writes the message to the log file with the TRACE level.

robot.api.logger.debug(msg, html=False)
    Writes the message to the log file with the DEBUG level.

robot.api.logger.info(msg, html=False, also_console=False)
    Writes the message to the log file with the INFO level.
    If also_console argument is set to True, the message is written both to the log file and to the console.

robot.api.logger.warn(msg, html=False)
    Writes the message to the log file with the WARN level.

robot.api.logger.console(msg, newline=True)
    Writes the message to the console.
    If the newline argument is True, a newline character is automatically added to the message.

class UserErrorHandler (name, error)
    Created if creating handlers fail – running raises DataError.
    The idea is not to raise DataError at processing time and prevent all tests in affected test case file from
    executing. Instead UserErrorHandler is created and if it is ever run DataError is raised then.
    type = 'error'
    init_keyword(varz)
    run(*args)

class BaseKeyword (name='', args=None, doc='', timeout='', type='kw')
Robot Framework Documentation, Release 2.7.7

```python
    passed
    serialize(serializer)
```

**libraries Module**

```python
class robot.common.libraries.BaseLibrary

    get_handler(name)
    has_handler(name)
```

**model Module**

```python
class robot.common.model.BaseTestSuite(name, source=None, parent=None):
    Bases: robot.common.model._TestAndSuiteHelper
    Base class for TestSuite used in runtime and but not anymore by robot.
    set_name(name)
    name
    id
    set_critical_tags(critical, non_critical)
    set_doc(doc)
    set_metadata(metalist)
    get_test_count()
    get_full_message()
        Returns suite’s message including statistics message
    get_stat_message()
    set_status()
        Sets status and statistics based on subsuite and test statuses.
        Can/should be used when statuses have been changed somehow.
    suite_teardown_failed(error=None, message=None)
    set_tags(tags)
    filter(suites=None, tests=None, includes=None, excludes=None, zero_tests_ok=False)
    filter_by_names(suites=None, tests=None, zero_tests_ok=False)
    filter_by_tags(includes=None, excludes=None, zero_tests_ok=False)
    set_runmode(runmode)
    set_options(settings)
    serialize(serializer)
    return_code
    longname
```
class robot.common.model.BaseTestCase
    Bases: robot.common.model._TestAndSuiteHelper
    id
    passed
    suite_teardown_failed
    set_criticality
    is_included
        Returns True if this test case is included but not excluded.
        If no ‘incl_tags’ are given all tests are considered to be included.
    serialize
    longname

statistics Module

class robot.common.statistics.Statistics
    serialize

class robot.common.statistics.Stat(name='')
    total
    add_stat
    add_test(test)
    fail_all()
    add_suite(suite)

class robot.common.statistics.CriticalStats
    Bases: robot.common.statistics.Stat
    add_stat
    add_suite
    add_test(test)
    fail_all()
    total

class robot.common.statistics.AllStats
    Bases: robot.common.statistics.Stat
    add_stat
    add_suite
    add_test(test)
    fail_all()
    total
class robot.common.statistics.SuiteStat(suite)
    Bases: robot.common.statistics.Stat
    type = 'suite'
    serialize(serializer)
    add_stat(other)
    add_suite(suite)
    add_test(test)
    fail_all()
    total

class robot.common.statistics.TagStat(name, doc='', links=[], critical=False, non_critical=False, combined='')
    Bases: robot.common.statistics.Stat
    type = 'tag'
    add_test(test)
    serialize(serializer)
    add_stat(other)
    add_suite(suite)
    fail_all()
    total

class robot.common.statistics.TotalStat(name, suite_stat)
    Bases: robot.common.statistics.Stat
    type = 'total'
    serialize(serializer)
    add_stat(other)
    add_suite(suite)
    add_test(test)
    fail_all()
    total

class robot.common.statistics.SuiteStatistics(suite, tag_stats, suite_stat_level=-1)
    serialize(serializer)

class robot.common.statistics.TagStatistics(include=None, exclude=None, combine=None, docs=None, links=None)
    add_test(test, critical)
    serialize(serializer)
    sort()

class robot.common.statistics.TotalStatistics(suite)
serialize(serializer)

class robot.common.statistics.TagStatInfo(docs, links)

    get_doc(tag)
    get_links(tag)

class robot.common.statistics.TagStatDoc(pattern, doc)

    matches(tag)

class robot.common.statistics.TagStatLink(pattern, link, title)

    matches(tag)
    get_link(tag)

conf Package

conf Package

Settings for both test execution and output processing.

settings Module

class robot.conf.settings.RobotSettings(options=None, log=True)
    Bases: robot.conf.settings._BaseSettings

    is_rebot_needed()
    get_rebot_datasource_and_settings()
    log
    output
    report
    split_log
    status_rc
    xunit

class robot.conf.settings.RebotSettings(options=None, log=True)
    Bases: robot.conf.settings._BaseSettings

    suite_config
    statistics_config
    log_config
    report_config
    log
    output
    report
htmldata Package

Generic functionality for writing output files in HTML format. robot.reporting, robot.libdoc and robot.testdoc use this package.

htmldata Module

class robot.htmldata.htmlfilewriter.HtmlFileWriter(output, model_writer)
    Bases: object
    write(template)

class robot.htmldata.htmlfilewriter.ModelWriter
    Bases: robot.htmldata.htmlfilewriter._Writer
    handles(line)
    write(line)

class robot.htmldata.htmlfilewriter.LineWriter(output)
    Bases: robot.htmldata.htmlfilewriter._Writer
    handles(line)
    write(line)

class robot.htmldata.htmlfilewriter.GeneratorWriter(html_writer)
    Bases: robot.htmldata.htmlfilewriter._Writer
    write(line)
    handles(line)

class robot.htmldata.htmlfilewriter.JsFileWriter(html_writer, base_dir)
    Bases: robot.htmldata.htmlfilewriter._InliningWriter
    write(line)
    handles(line)

class robot.htmldata.htmlfilewriter.CssFileWriter(html_writer, base_dir)
    Bases: robot.htmldata.htmlfilewriter._InliningWriter
    write(line)
    handles(line)
**jartemplate Module**

**jsonwriter Module**

```python
class robot.htmldata.jsonwriter.JsonWriter(output, separator='')
Bases: object
    write_json (prefix, data, postfix=';n', mapping=Non, separator=True)
    write (string, postfix=';n', separator=True)

class robot.htmldata.jsonwriter.JsonDumper(output)
Bases: object
    dump (data, mapping=None)
    write (data)

class robot.htmldata.jsonwriter.StringDumper(jsondumper)
Bases: robot.htmldata.jsonwriter._Dumper
    dump (data, mapping)
    handles (data, mapping)

class robot.htmldata.jsonwriter.IntegerDumper(jsondumper)
Bases: robot.htmldata.jsonwriter._Dumper
    dump (data, mapping)
    handles (data, mapping)

class robot.htmldata.jsonwriter.DictDumper(jsondumper)
Bases: robot.htmldata.jsonwriter._Dumper
    dump (data, mapping)
    handles (data, mapping)

class robot.htmldata.jsonwriter.TupleListDumper(jsondumper)
Bases: robot.htmldata.jsonwriter._Dumper
    dump (data, mapping)
    handles (data, mapping)

class robot.htmldata.jsonwriter.MappingDumper(jsondumper)
Bases: robot.htmldata.jsonwriter._Dumper
    handles (data, mapping)
    dump (data, mapping)

class robot.htmldata.jsonwriter.NoneDumper(jsondumper)
Bases: robot.htmldata.jsonwriter._Dumper
    handles (data, mapping)
    dump (data, mapping)
```

**normaltemplate Module**

```python
class robot.htmldata.normaltemplate.HtmlTemplate(filename)
Bases: object
```
template Module

libdopckg Package

libdopckg Package

Implements libdoc tool.
For programmatic entry point, see robot.libdoc.
This package is considered stable.

robot.libdopckg.LibraryDocumentation(library_or_resource, name=None, version=None, doc_format=None)

builder Module

robot.libdopckg.builder.JavaDocBuilder()
robot.libdopckg.builder.DocumentationBuilder(library_or_resource)

consoleviewer Module

class robot.libdopckg.consoleviewer.ConsoleViewer(libdoc)
    Bases: object
        classmethod handles(command)
        classmethod validate_command(command, args)
        view(command, *args)
        list(*patterns)
        show(*names)
        version()

class robot.libdopckg.consoleviewer.KeywordMatcher(libdoc)
    Bases: object
        search(patterns)

htmlwriter Module

class robot.libdopckg.htmlwriter.LibdocHtmlWriter
    Bases: object
        write(libdoc, output)

class robot.libdopckg.htmlwriter.LibdocModelWriter(output, libdoc)
    Bases: robot.htmldata.htmlfilewriter.ModelWriter
        write(line)
        write_data()
        handles(line)
class robot.libdocpkg.htmlwriter.JsonConverter (doc_formatter)
    Bases: object
    convert (libdoc)

class robot.libdocpkg.htmlwriter.DocFormatter (keywords, introduction, doc_format='ROBOT')
    Bases: object
    html (doc, intro=False)

class robot.libdocpkg.htmlwriter.DocToHtml (format)
    Bases: object

javabuilder Module

class robot.libdocpkg.javabuilder.JavaDocBuilder
    Bases: object
    build (path)

robot.libdocpkg.javabuilder.ClassDoc (path)
    Process the given Java source file and return ClassDoc instance.
    Processing is done using com.sun.tools.javadoc APIs. The usage has been figured out from sources at http://www.java2s.com/Open-Source/Java-Document/JDK-Modules-com.sun/tools/com.sun.tools.javadoc.htm
    Returned object implements com.sun.javadoc.ClassDoc interface, see http://java.sun.com/j2se/1.4.2/docs/tooldocs/javadoc/doclet/

model Module

class robot.libdocpkg.model.LibraryDoc (name='', doc='', version=' ', type='library', scope=' ', named_args=False, doc_format='')
    Bases: object
    scope
    doc_format
    keywords
    save (output=None, format='HTML')

class robot.libdocpkg.model.KeywordDoc (name=' ', args=None, doc=' ')
    Bases: object
    shortdoc

output Module

class robot.libdocpkg.output.LibdocOutput (output_path, format)
    Bases: object

robotbuilder Module

class robot.libdocpkg.robotbuilder.LibraryDocBuilder
    Bases: object
build (library)

class robot.libdocpkg.robotbuilder.ResourceDocBuilder
    Bases: object
    build (path)

class robot.libdocpkg.robotbuilder.KeywordDocBuilder
    Bases: object
    build_keywords (lib)
    build_keyword (kw)

specbuilder Module

class robot.libdocpkg.specbuilder.SpecDocBuilder
    Bases: object
    build (path)

writer Module

robot.libdocpkg.writer.LibdocWriter (format=None)

xmlwriter Module

class robot.libdocpkg.xmlwriter.LibdocXmlWriter
    Bases: object
    write (libdoc, outfile)

libraries Package

libraries Package

Contains Robot Framework standard test libraries.

BuiltIn Module

class robot.libraries.BuiltIn.BuiltIn
    An always available standard library with often needed keywords.

    BuiltIn is Robot Framework’s standard library that provides a set of generic keywords needed often. It is imported automatically and thus always available. The provided keywords can be used, for example, for verifications (e.g. Should Be Equal, Should Contain), conversions (e.g. Convert To Integer) and for various other purposes (e.g. Log, Sleep, Run Keyword If, Set Global Variable).

    ROBOT_LIBRARY_SCOPE = ‘GLOBAL’
    ROBOT_LIBRARY_VERSION = ‘2.7.7’
**call_method** *(object, method_name, *args)*

Calls the named method of the given object with the provided arguments.

The possible return value from the method is returned and can be assigned to a variable. Keyword fails both if the object does not have a method with the given name or if executing the method raises an exception.

Examples:

```
| Call Method | ${hashtable} | put | myname | myvalue |
| Should Not Be True | ${{isempty}} = l | Call Method | ${{hashtable}} | isEmpty |
| Should Be Equal | ${{value}} | myvalue |
```

**catenate** *(items)*

Catenates the given items together and returns the resulted string.

By default, items are catenated with spaces, but if the first item contains the string 'SEPARATOR=<sep>', the separator '<<sep>>' is used. Items are converted into strings when necessary.

Examples:

```
| ${{str1}} = | Catenate | Hello | world |
| ${{str2}} = | Catenate | SEPARATOR=— | Hello | world |
| ${{str3}} = | Catenate | SEPARATOR= | Hello | world | => | ${{str1}} = 'Hello world' |
```

**comment** *(messages)*

Displays the given messages in the log file as keyword arguments.

This keyword does nothing with the arguments it receives, but as they are visible in the log, this keyword can be used to display simple messages. Given arguments are ignored so thoroughly that they can even contain non-existing variables. If you are interested about variable values, you can use the *Log* or *Log Many* keywords.

**convert_to_binary** *(item, base=None, prefix=None, length=None)*

Converts the given item to a binary string.

The *item*, with an optional *base*, is first converted to an integer using *Convert To Integer* internally. After that it is converted to a binary number (base 2) represented as a string such as '1011'.

The returned value can contain an optional *prefix* and can be required to be of minimum *length* (excluding the prefix and a possible minus sign). If the value is initially shorter than the required length, it is padded with zeros.

Examples:

```
| ${{result}} = | Convert To Binary | 10 | # Result is 1010 |
| ${{result}} = | Convert To Binary | F | base=16 | prefix=0b | # Result is 0b1111 |
| ${{result}} = | Convert To Binary | -2 | prefix=B | length=4 | # Result is -B0010 |
```

This keyword was added in Robot Framework 2.6. See also *Convert To Integer*, *Convert To Octal* and *Convert To Hex*.

**convert_to_boolean** *(item)*

Converts the given item to Boolean true or false.

Handles strings 'True' and 'False' (case-insensitive) as expected, otherwise returns item’s truth value using Python’s ‘bool’ method. For more information about truth values, see [http://docs.python.org/lib/truth.html](http://docs.python.org/lib/truth.html).

**convert_to_hex** *(item, base=None, prefix=None, length=None, lowercase=False)*

Converts the given item to a hexadecimal string.

The *item*, with an optional *base*, is first converted to an integer using *Convert To Integer* internally. After that it is converted to a hexadecimal number (base 16) represented as a string such as ‘FF0A’.

The returned value can contain an optional *prefix* and can be required to be of minimum *length* (excluding the prefix and a possible minus sign). If the value is initially shorter than the required length, it is padded with zeros.

By default the value is returned as an upper case string, but giving any non-empty value to the *lowercase* argument turns the value (but not the prefix) to lower case.
Robot Framework Documentation, Release 2.7.7

Examples:  |
| \${result} = | Convert To Hex | 255 | \# Result is FF |
| \${result} = | Convert To Hex | -10 | prefix=0x | length=2 | \# Result is -0x0A |
| \${result} = | Convert To Hex | 255 | prefix=X | lowercase=yes | \# Result is Xff |

This keyword was added in Robot Framework 2.6. See also Convert To Integer, Convert To Binary and Convert To Octal.

**convert_to_integer** (*item, base=None*)

Converts the given item to an integer number.

If the given item is a string, it is by default expected to be an integer in base 10. Starting from Robot Framework 2.6 there are two ways to convert from other bases:

1. Give base explicitly to the keyword as `base` argument.
2. Prefix the given string with the base so that `0b` means binary (base 2), `0o` means octal (base 8), and `0x` means hex (base 16). The prefix is considered only when `base` argument is not given and may itself be prefixed with a plus or minus sign.

The syntax is case-insensitive and possible spaces are ignored.

Examples:  |
| \${result} = | Convert To Integer | 100 | \# Result is 100 |
| \${result} = | Convert To Integer | FF AA | 16 | \# Result is 65450 |
| \${result} = | Convert To Integer | 100 | 8 | \# Result is 64 |
| \${result} = | Convert To Integer | -100 | 2 | \# Result is -4 |
| \${result} = | Convert To Integer | 0b100 | \# Result is 4 |
| \${result} = | Convert To Integer | -0x100 | \# Result is -256 |

See also Convert To Number, Convert To Binary, Convert To Octal and Convert To Hex.

**convert_to_number** (*item, precision=None*)

Converts the given item to a floating point number.

If the optional `precision` is positive or zero, the returned number is rounded to that number of decimal digits. Negative precision means that the number is rounded to the closest multiple of 10 to the power of the absolute precision. The support for precision was added in Robot Framework 2.6.

Examples:  |
| \${result} = | Convert To Number | 42.512 | \# Result is 42.512 |
| \${result} = | Convert To Number | 42.512 | 1 | \# Result is 42.5 |
| \${result} = | Convert To Number | 42.512 | 0 | \# Result is 43.0 |
| \${result} = | Convert To Number | 42.512 | -1 | \# Result is 40.0 |

Notice that machines generally cannot store floating point numbers accurately. This may cause surprises with these numbers in general and also when they are rounded. For more information see, for example, this floating point arithmetic tutorial: [http://docs.python.org/tutorial/floatingpoint.html](http://docs.python.org/tutorial/floatingpoint.html)

If you need an integer number, use Convert To Integer instead.

**convert_to_octal** (*item, base=None, prefix=None, length=None*)

Converts the given item to an octal string.

The `item`, with an optional `base`, is first converted to an integer using Convert To Integer internally. After that it is converted to an octal number (base 8) represented as a string such as ‘775’.

The returned value can contain an optional `prefix` and can be required to be of minimum `length` (excluding the prefix and a possible minus sign). If the value is initially shorter than the required length, it is padded with zeros.

Examples:  |
| \${result} = | Convert To Octal | 10 | \# Result is 12 |
| \${result} = | Convert To Octal | -F | base=16 | prefix=0 | \# Result is -017 |
| \${result} = | Convert To Octal | 16 | prefix=oct | length=4 | \# Result is oct0020 |

This keyword was added in Robot Framework 2.6. See also Convert To Integer, Convert To Binary and Convert To Hex.
**convert_to_string** (*item*)

Converts the given item to a Unicode string.

Uses '__unicode__' or '__str__' method with Python objects and 'toString' with Java objects.

Use **Encode String To Bytes** and **Decode Bytes To String** keywords in **String** library if you need to convert between Unicode and byte strings.

**create_list** (*items*)

Returns a list containing given items.

The returned list can be assigned both to ${scalar} and @ {list} variables. The earlier can be used e.g. with Java keywords expecting an array as an argument.

Examples:

| @{list} = | Create List | a | b | c |
| ${scalar} = | Create List | a | b | c |
| ${ints} = | Create List | ${1} | ${2} | ${3} |

**evaluate** (**expression**, **modules=**None)

Evaluates the given expression in Python and returns the results.

The **modules** argument can be used to specify a comma separated list of Python modules to be imported and added to the namespace of the evaluated expression.

Examples (expecting $\{result\}$ is 3.14):

| ${status} = | Evaluate | 0 < $\{result\} < 10 | |
| ${down} = | Evaluate | int($\{result\}) | |
| ${up} = | Evaluate | math.ceil($\{result\}) | |
| ${random} = | Evaluate | random.randint(0, sys.maxint) | |

Notice that instead of creating complicated expressions, it is recommended to move the logic into a test library.

**exit_for_loop** ()

Immediately stops executing the enclosing for loop.

This keyword can be used directly in a for loop or in a keyword that the for loop uses. In both cases the test execution continues after the for loop. If executed outside of a for loop, the test fails.

Example:

| :FOR | ${var} | IN | @{SOME LIST} | |
| Run Keyword If | '${var}' == 'EXIT' | Exit For Loop |
| Do Something | ${var} |

New in Robot Framework 2.5.2.

**fail** (**msg=**None, *tags*)

Fails the test with the given message and optionally alters its tags.

The error message is specified using the optional **msg** argument.

Starting from Robot Framework 2.7.4, it is possible to modify tags of the current test case by passing tags after the message. Tags starting with a hyphen (e.g. -regression) are removed and others added. Tags are modified using **Set Tags** and **Remove Tags** internally, and the semantics setting and removing them are the same as with these keywords.

Examples:

| Fail | Keyword not ready | # Fails with the given message. |
| Fail | Keyword not ready | # Fails and adds 'not-ready' tag. |
| Fail | OS not supported | # Removes tag 'regression'. |
| Fail | My message | -old | new | # Adds tag 'new' and removes 'old'. |
| Fail | My message | # Removes all tags starting with 't' except the newly added 'tag'. |

See **Fatal Error** if you need to stop the whole test execution.

**fatal_error** (**msg=**None)

Stops the whole test execution.

The test or suite where this keyword is used fails with the provided message, and subsequent tests fail with a canned message. Possible tear downs will nevertheless be executed.
See *Fail* if you only want to stop one test case unconditionally.

**get_count** (*item1*, *item2*)

Returns and logs how many times *item2* is found from *item1*.

This keyword works with Python strings and lists and all objects that either have `count` method or can be converted to Python lists.

Example: | ${count} = | Get Count | ${some item} | interesting value | | Should Be True | 5 < ${count} < 10 |

**get_length** (*item*)

Returns and logs the length of the given item.

The item can be anything that has a length, for example, a string, a list, or a mapping. The keyword first tries to get the length with the Python function `len`, which calls the item’s `_len_` method internally. If that fails, the keyword tries to call the item’s possible `length` and `size` methods directly. The final attempt is trying to get the value of the item’s `length` attribute. If all these attempts are unsuccessful, the keyword fails.

It is possible to use this keyword also with list variables (e.g. `@{LIST}`), but you need to use them as scalars (e.g. `$\{LIST\}$`).

**get_library_instance** (*name*)

Returns the currently active instance of the specified test library.

This keyword makes it easy for test libraries to interact with other test libraries that have state. This is illustrated by the Python example below:

```python
from robot.libraries.BuiltIn import BuiltIn

def title_should_start_with(expected):
    seleniumlib = BuiltIn().get_library_instance('SeleniumLibrary')
    title = seleniumlib.get_title()
    if not title.startswith(expected):
        raise AssertionError("Title '%s' did not start with '%s'",
                          % (title, expected))
```

It is also possible to use this keyword in the test data and pass the returned library instance to another keyword. If a library is imported with a custom name, the *name* used to get the instance must be that name and not the original library name.

**get_time** (*format* = `timestamp`, *time_* = `NOW`)

Returns the given time in the requested format.

How time is returned is determined based on the given *format* string as follows. Note that all checks are case-insensitive.

1. If *format* contains the word ‘epoch’, the time is returned in seconds after the UNIX epoch (1970-01-01 00:00:00 UTC). The return value is always an integer.
2. If *format* contains any of the words ‘year’, ‘month’, ‘day’, ‘hour’, ‘min’, or ‘sec’, only the selected parts are returned. The order of the returned parts is always the one in the previous sentence and the order of words in *format* is not significant. The parts are returned as zero-padded strings (e.g. May -> ‘05’).
3. Otherwise (and by default) the time is returned as a timestamp string in the format ‘2006-02-24 15:08:31’.

By default this keyword returns the current local time, but that can be altered using the `time` argument as explained below. Note that all checks involving strings are case-insensitive.

1. If `time` is a number, or a string that can be converted to a number, it is interpreted as seconds since the UNIX epoch. This documentation was originally written about 1177654467 seconds after the epoch.

2. If `time` is a timestamp, that time will be used. Valid timestamp formats are ‘YYYY-MM-DD hh:mm:ss’ and ‘YYYYMMDD hhmmss’.

3. If `time` is equal to ‘NOW’ (default), the current local time is used. This time is got using Python’s `time()` function.

4. If `time` is equal to ‘UTC’, the current time in [http://en.wikipedia.org/wiki/Coordinated_Universal_Time|UTC] is used. This time is got using `time.time()` + `time.altzone` in Python.

5. If `time` is in the format like ‘NOW - 1 day’ or ‘UTC + 1 hour 30 min’, the current local/UTC time plus/minus the time specified with the time string is used. The time string format is described in an appendix of Robot Framework User Guide.

Examples (expecting the current local time is 2006-03-29 15:06:21):

| ${time} = Get Time |
| | => | ${time} = ‘2006-03-29 15:06:21’ |

| ${time} = Get Time | epoch |
| | => | ${time} = 1177654467 |

| ${year} = Get Time | return year |
| | => | ${year} = ‘2006’ |

| @{time} = Get Time | year month day hour minute second |
| | => | @{time} = ['2006', '03', '29', '15', '06', '21'] |

| ${y} = Get Time | seconds and year |
| | => | ${y} = ‘2006’ |

Examples (expecting the current local time is 2006-03-29 15:06:21 and UTC time is 2006-03-29 12:06:21):

| ${time} = Get Time | epoch seconds |
| | => | ${time} = 1177654467 |

| ${year} = Get Time | year |
| | => | ${year} = ‘2006’ |

| @{time} = Get Time | hour minute second |
| | => | @{time} = ['16', '08', '24'] |

Support for UTC time was added in Robot Framework 2.7.5 but it did not work correctly until 2.7.7.

`get_variable_value(name, default=None)`

Returns variable value or `default` if the variable does not exist.

The name of the variable can be given either as a normal variable name (e.g. `$NAME`) or in escaped format (e.g. `$\{NAME\}`). Notice that the former has some limitations explained in `Set Suite Variable`.

Examples:

| `$x` = Get Variable Value |
| | => | `$x` = None |

| `$y` = Get Variable Value |
| | => | `$y` = None |

| `$z` = Get Variable Value |
| | => | `$z` = None |

This keyword was added in Robot Framework 2.6. See `Set Variable If` for another keyword to set variables dynamically.

`get_variables()`

Returns a dictionary containing all variables in the current scope.

Variables are returned as a special dictionary that allows accessing variables in space, case, and underscore insensitive manner similarly as accessing variables in the test data. This dictionary supports all same operations as normal Python dictionaries and, for example, Collections library can be used to access or modify it. Modifying the returned dictionary has no effect on the variables available in the current scope.
Example:  | ${example_variable} = | Set Variable | example value | | ${variables} = | Get Variables | | | Dictionary Should Contain Key | ${variables} | | Dictionary Should Contain Key | $ {variables} | | | Dictionary Should Contain Key | $ {ExampleVariable} | | Set To Dictionary | ${variables} | | | Variable Should Not Exist | ${name} | | | Note: Prior to Robot Framework 2.7.4 variables were returned as a custom object that did not support all dictionary methods.

`import_library(name, *args)`
Imports a library with the given name and optional arguments.

This functionality allows dynamic importing of libraries while tests are running. That may be necessary, if the library itself is dynamic and not yet available when test data is processed. In a normal case, libraries should be imported using the Library setting in the Setting table.

This keyword supports importing libraries both using library names and physical paths. When path are used, they must be given in absolute format. Forward slashes can be used as path separators in all operating systems. It is possible to use arguments as well as to give a custom name with ‘WITH NAME’ syntax. For more information about importing libraries, see Robot Framework User Guide.

Examples:
- Import Library | MyLibrary
- Import Library | ${CURDIR}/Library.py | some | args
- Import Library | ${CURDIR}/../libs/Lib.java | arg | WITH NAME | JavaLib

`import_resource(path)`
Imports a resource file with the given path.

Resources imported with this keyword are set into the test suite scope similarly when importing them in the Setting table using the Resource setting.

The given path must be absolute. Forward slashes can be used as path separator regardless the operating system.

Examples:
- Import Resource | ${CURDIR}/resource.txt
- Import Resource | ${CURDIR}/../resources/resource.html

`import_variables(path, *args)`
Imports a variable file with the given path and optional arguments.

Variables imported with this keyword are set into the test suite scope similarly when importing them in the Setting table using the Variables setting. These variables override possible existing variables with the same names and this functionality can thus be used to import new variables, e.g. for each test in a test suite.

The given path must be absolute. Forward slashes can be used as path separator regardless the operating system.

Examples:
- Import Variables | ${CURDIR}/variables.py
- Import Variables | ${CURDIR}/../vars/env.py

New in Robot Framework 2.5.4.

`keyword_should_exist(name, msg=None)`
Fails unless the given keyword exists in the current scope.

Fails also if there are more than one keywords with the same name. Works both with the short name (e.g. `Log`) and the full name (e.g. `BuiltIn.Log`).

The default error message can be overridden with the `msg` argument.

New in Robot Framework 2.6. See also `Variable Should Exist`.

`length_should_be(item, length, msg=None)`
Verifies that the length of the given item is correct.
The length of the item is got using the Get Length keyword. The default error message can be overridden with the msg argument.

```robot
log(message, level='INFO')
```
Logs the given message with the given level.

Valid levels are TRACE, DEBUG, INFO (default), HTML and WARN.

The HTML level is special because it allows writing messages without HTML code in them being escaped. For example, logging a message `<img src="image.png">` using the HTML level creates an image, but with other levels the message would be that exact string. Notice that invalid HTML can easily corrupt the whole log file so this feature should be used with care. The actual log level used for HTML messages is INFO.

Messages logged with the WARN level will be visible also in the console and in the Test Execution Errors section in the log file.

```robot
log_many(*messages)
```
Logs the given messages as separate entries with the INFO level.

```robot
log_variables(level='INFO')
```
Logs all variables in the current scope with given log level.

```robot
no_operation()
```
Does absolutely nothing.

```robot
regexp_escape(*patterns)
```
Returns each argument string escaped for use as a regular expression.

This keyword can be used to escape strings to be used with Should Match Regexp and Should Not Match Regexp keywords.

Escaping is done with Python’s re.escape() function.

Examples: | ${escaped} = | Regexp Escape | ${original} | | @{strings} = | Regexp Escape | @{strings} |

```robot
remove_tags(*tags)
```
Removes given tags from the current test or all tests in a suite.

Tags can be given exactly or using a pattern where ‘*’ matches anything and ‘?’ matches one character.

This keyword can affect either one test case or all test cases in a test suite similarly as Set Tags keyword.

The current tags are available as a built-in variable @TEST TAGS@.

Example: | Remove Tags | mytag | something-* | ?ython |

See Set Tags if you want to add certain tags and Fail if you want to fail the test case after setting and/or removing tags.

```robot
repeat_keyword(times, name, *args)
```
Executes the specified keyword multiple times.

`name` and `args` define the keyword that is executed similarly as with Run Keyword, and `times` specifies how many the keyword should be executed. `times` can be given as an integer or as a string that can be converted to an integer. It can also have postfix ‘times’ or ‘x’ (case and space insensitive) to make the expression easier to read.

If `times` is zero or negative, the keyword is not executed at all. This keyword fails immediately if any of the execution rounds fails.

Examples: | Repeat Keyword | 5 times | Goto Previous Page | | Repeat Keyword | ${var} | Some Keyword | arg1 | arg2 |
replace_variables(text)
Replaces variables in the given text with their current values.

If the text contains undefined variables, this keyword fails. If the given text contains only a single variable, its value is returned as-is and it can be any object. Otherwise this keyword always returns a string.

Example:
The file ‘template.txt’ contains ‘Hello ${NAME}!’ and variable ‘${NAME}’ has the value ‘Robot’.

```
${template} = | Get File | ${CURDIR}/template.txt |
${message} = | Replace Variables | ${template} |
Should Be Equal | ${message} | Hello Robot! |
```

run_keyword(name, *args)
Executes the given keyword with the given arguments.

Because the name of the keyword to execute is given as an argument, it can be a variable and thus set dynamically, e.g. from a return value of another keyword or from the command line.

run_keyword_and_continue_on_failure(name, *args)
Runs the keyword and continues execution even if a failure occurs.

The keyword name and arguments work as with Run Keyword.

Example: | Run Keyword And Continue On Failure | Fail | This is a stupid example | | Log | This keyword is executed |

This keyword was added in Robot Framework 2.5. The execution is not continued if the failure is caused by invalid syntax, timeout, or fatal exception.

run_keyword_and_expect_error(expected_error, name, *args)
Runs the keyword and checks that the expected error occurred.

The expected error must be given in the same format as in Robot Framework reports. It can be a pattern containing characters ‘?’, which matches to any single character and ‘*’, which matches to any number of any characters. name and *args have same semantics as with Run Keyword.

If the expected error occurs, the error message is returned and it can be further processed/tested, if needed. If there is no error, or the error does not match the expected error, this keyword fails.

Examples: | Run Keyword And Expect Error | My error | Some Keyword | arg1 | arg2 | $msg = | Run Keyword And Expect Error | * | My KW | Should Start With | $msg | Once upon a time in |

Starting from Robot Framework 2.5 errors caused by invalid syntax, timeouts, or fatal exceptions are not caught by this keyword.

run_keyword_and_ignore_error(name, *args)
Runs the given keyword with the given arguments and ignores possible error.

This keyword returns two values, so that the first is either ‘PASS’ or ‘FAIL’, depending on the status of the executed keyword. The second value is either the return value of the keyword or the received error message. See Run Keyword And Return Status If you are only interested in the execution status.

The keyword name and arguments work as in Run Keyword. See Run Keyword If for a usage example.

Starting from Robot Framework 2.5 errors caused by invalid syntax, timeouts, or fatal exceptions are not caught by this keyword.
run_keyword_and_return_status (name, *args)
Runs the given keyword with given arguments and returns the status as a Boolean value.

This keyword returns True if the keyword that is executed succeeds and False if it fails. This is useful, for example, in combination with Run Keyword If. If you are interested in the error message or return value, use Run Keyword And Ignore Error instead.

The keyword name and arguments work as in Run Keyword.

Example: | ${passed} = | Run Keyword And Return Status | Keyword | args | Run Keyword If | | Another keyword |

New in Robot Framework 2.7.6.

run_keyword_if (condition, name, *args)
Runs the given keyword with the given arguments, if condition is true.

The given condition is evaluated similarly as with Should Be True keyword, and name and *args have same semantics as with Run Keyword.

Example, a simple if/else construct: | ${status} | ${value} = | Run Keyword And Ignore Error | My Keyword | Run Keyword If | `${status}` == 'PASS' | Some Action | arg | Run Keyword Unless | `${status}` == 'PASS' | Another Action |

In this example, only either Some Action or Another Action is executed, based on the status of My Keyword. Instead of Run Keyword And Ignore Error you can also use Run Keyword And Return Status.

Starting from Robot version 2.7.4, this keyword supports also optional ELSE and ELSE IF branches. Both of these are defined in *args and must use exactly format ELSE or ELSE IF, respectively. ELSE branches must contain first the name of the keyword to execute and then its possible arguments. ELSE IF branches must first contain a condition, like the first argument to this keyword, and then the keyword to execute and its possible arguments. It is possible to have ELSE branch after ELSE IF and to have multiple ELSE IF branches.

Given previous example, if/else construct can also be created like this: | ${status} | ${value} = | Run Keyword And Ignore Error | My Keyword | Run Keyword If | `${status}` == 'PASS' | Some Action | arg | ELSE | Another Action |

Using ELSE and/or ELSE IF branches is especially handy if you are interested in the return value. This is illustrated by the example below that also demonstrates using ELSE IF and ELSE together:

| ${result} = | Run Keyword If | ${rc} == 0 | Zero return value |
| ... | ELSE IF | 0 < ${rc} < 42 | Normal return value |
| ... | ELSE IF | ${rc} < 0 | Negative return value | ${rc} | arg2 |
| ... | ELSE | Abnormal return value | ${rc} |

Notice that ELSE and ELSE IF control arguments must be used explicitly and thus cannot come from variables. If you need to use literal ELSE and ELSE IF strings as arguments, you can either use variables or escape them with a backslash like ELSE and ELSE IF.

run_keyword_if_all_critical_tests_passed (name, *args)
Runs the given keyword with the given arguments, if all critical tests passed.

This keyword can only be used in suite teardown. Trying to use it in any other place will result in an error.

Otherwise, this keyword works exactly like Run Keyword, see its documentation for more details.

run_keyword_if_all_tests_passed (name, *args)
Runs the given keyword with the given arguments, if all tests passed.
This keyword can only be used in a suite teardown. Trying to use it anywhere else results in an error. Otherwise, this keyword works exactly like Run Keyword, see its documentation for more details.

**run_keyword_if_any_critical_tests_failed**(name, *args)
Runs the given keyword with the given arguments, if any critical tests failed.
This keyword can only be used in a suite teardown. Trying to use it anywhere else results in an error. Otherwise, this keyword works exactly like Run Keyword, see its documentation for more details.

**run_keyword_if_any_tests_failed**(name, *args)
Runs the given keyword with the given arguments, if one or more tests failed.
This keyword can only be used in a suite teardown. Trying to use it anywhere else results in an error. Otherwise, this keyword works exactly like Run Keyword, see its documentation for more details.

**run_keyword_if_test_failed**(name, *args)
Runs the given keyword with the given arguments, if the test failed.
This keyword can only be used in a suite teardown. Trying to use it anywhere else results in an error. Otherwise, this keyword works exactly like Run Keyword, see its documentation for more details.

**run_keyword_if_test_passed**(name, *args)
Runs the given keyword with the given arguments, if the test passed.
This keyword can only be used in a test teardown. Trying to use it anywhere else results in an error. Otherwise, this keyword works exactly like Run Keyword, see its documentation for more details.

**run_keyword_if_timeout_occurred**(name, *args)
Runs the given keyword if either a test or a keyword timeout has occurred.
This keyword can only be used in a test teardown. Trying to use it anywhere else results in an error. Otherwise, this keyword works exactly like Run Keyword, see its documentation for more details.

Available in Robot Framework 2.5 and newer.

**run_keyword_unless**(condition, name, *args)
Runs the given keyword with the given arguments, if condition is false.
See Run Keyword If for more information and an example.

**run_keywords**( *keywords)
Executes all the given keywords in a sequence.
This keyword is mainly useful in setups and teardowns when they need to take care of multiple actions and creating a new higher level user keyword would be an overkill.

Examples: 
| Run Keywords | Initialize database | Start servers | Clear logs |
| Run Keywords | ${KW 1} | ${KW 2} |
| Run Keywords | ${KEYWORDS} |

In this example, we call Run Keywords with three different combination of arguments. Keyword names and arguments can come from variables, as demonstrated in the second and third rows.

Starting from Robot Framework 2.7.6, keywords can also be run with arguments using upper case AND as a separator between keywords. The keywords are executed so that the first argument is the first keyword and proceeding arguments until the next AND are arguments to it. First argument after the first AND is the second keyword and proceeding arguments until the next AND are its arguments. And so on.

Examples: 
| Run Keywords | Initialize database | db1 | AND | Start servers | server1 | server2 |
| Run Keywords | Initialize database | ${DB NAME} | AND | Start servers | @ {SERVERS} | AND | Clear logs |
| Run Keywords | ${KW} | AND | @ {KW WITH ARGS} |
Notice that the AND control argument must be used explicitly and thus cannot itself come from a variable. If you need to use literal AND string as argument, you can either use variables or escape it with a backslash like AND.

**set_global_variable** (*name*, *values*)

Makes a variable available globally in all tests and suites.

Variables set with this keyword are globally available in all test cases and suites executed after setting them. Setting variables with this keyword thus has the same effect as creating from the command line using the options ‘--variable’ or ‘--variablefile’. Because this keyword can change variables everywhere, it should be used with care.

See *Set Suite Variable* for more information and examples.

**set_library_search_order** (*libraries*)

Sets the resolution order to use when a name matches multiple keywords.

The library search order is used to resolve conflicts when a keyword name in the test data matches multiple keywords. The first library (or resource, see below) containing the keyword is selected and that keyword implementation used. If the keyword is not found from any library (or resource), test executing fails the same way as when the search order is not set.

When this keyword is used, there is no need to use the long *LibraryName.Keyword Name* notation. For example, instead of having

```
MyLibrary.Keyword | arg |
MyLibrary.Another Keyword |
MyLibrary.Keyword | xxx |
```

you can have

```
Set Library Search Order | MyLibrary |
Keyword | arg |
Another Keyword |
Keyword | xxx |
```

Starting from Robot Framework 2.6.2 this keyword can be used also to set the order of keywords in different resource files. In this case resource names must be given without paths or extensions like:

```
Set Library Search Order | resource | another_resource |
```

**NOTE:** - The search order is valid only in the suite where this keywords is used. - Keywords in resources always have higher priority than

keywords in libraries regardless the search order.

• The old order is returned and can be used to reset the search order later.

• Starting from RF 2.6.2, library and resource names in the search order are both case and space insensitive.
**set_log_level** *(level)*
Sets the log threshold to the specified level and returns the old level.

Messages below the level will not logged. The default logging level is INFO, but it can be overridden with the command line option `--loglevel`.

The available levels: TRACE, DEBUG, INFO (default), WARN and NONE (no logging).

**set_suite_documentation** *(doc, append=False, top=False)*
Sets documentation for the current test suite.

By default the possible existing documentation is overwritten, but this can be changed using the optional *append* argument similarly as with *Set Test Message* keyword.

This keyword sets the documentation of the current suite by default. If the optional *top* argument is given any value considered *true* in Python, for example, any non-empty string, the documentation of the top level suite is altered instead.

The documentation of the current suite is available as a built-in variable `SUITE DOCUMENTATION`.

New in Robot Framework 2.7. Support for *append* and *top* were added in 2.7.7.

**set_suite_metadata** *(name, value, append=False, top=False)*
Sets metadata for the current test suite.

By default possible existing metadata values are overwritten, but this can be changed using the optional *append* argument similarly as with *Set Test Message* keyword.

This keyword sets the metadata of the current suite by default. If the optional *top* argument is given any value considered *true* in Python, for example, any non-empty string, the metadata of the top level suite is altered instead.

The metadata of the current suite is available as a built-in variable `SUITE METADATA` in a Python dictionary. Notice that modifying this variable directly has no effect on the actual metadata the suite has.

New in Robot Framework 2.7.4. Support for *append* and *top* were added in 2.7.7.

**set_suite_variable** *(name, *values)*
Makes a variable available everywhere within the scope of the current suite.

Variables set with this keyword are available everywhere within the scope of the currently executed test suite. Setting variables with this keyword thus has the same effect as creating them using the Variable table in the test data file or importing them from variable files. Other test suites, including possible child test suites, will not see variables set with this keyword.

The name of the variable can be given either as a normal variable name (e.g. `$NAME`) or in escaped format as `@NAME` or `@{NAME}`.

If a variable already exists within the new scope, its value will be overwritten. Otherwise a new variable is created. If a variable already exists within the current scope, the value can be left empty and the variable within the new scope gets the value within the current scope.

Examples: | Set Suite Variable | ${GREET} | Hello, world! |
| Set Suite Variable | @{LIST} | First item |
| Second item | | $ID = | Get ID |
| Set Suite Variable | $ID |

To override an existing value with an empty value, use built-in variables `$EMPTY` or `@{EMPTY}:

Set Suite Variable | ${GREET} | $EMPTY |
Set Suite Variable | @{LIST} | @{EMPTY} | # New in RF 2.7.4 |
NOTE: If the variable has value which itself is a variable (escaped or not), you must always use the escaped format to reset the variable:

Example: | ${NAME} = | Set Variable | ${var} | | Set Suite Variable | ${NAME} | value | # Sets variable
${var} | | Set Suite Variable | ${NAME} | value | # Sets variable ${NAME} |

This limitation applies also to Set Test/Suite/Global Variable, Variable Should (Not) Exist, and Get Variable Value keywords.

**set_tags** (*tags*)

Adds given *tags* for the current test or all tests in a suite.

When this keyword is used inside a test case, that test gets the specified tags and other tests are not affected.

If this keyword is used in a suite setup, all test cases in that suite, recursively, gets the given tags. It is a failure to use this keyword in a suite teardown.

The current tags are available as a built-in variable `@{TEST TAGS}`.

See **Remove Tags** if you want to remove certain tags and **Fail** if you want to fail the test case after setting and/or removing tags.

**set_test_documentation** (*doc, append=False*)

Sets documentation for the current test case.

By default the possible existing documentation is overwritten, but this can be changed using the optional `append` argument similarly as with **Set Test Message** keyword.

The current test documentation is available as a built-in variable `$[TEST DOCUMENTATION]`. This keyword can not be used in suite setup or suite teardown.

New in Robot Framework 2.7. Support for `append` was added in 2.7.7.

**set_test_message** (*message, append=False*)

Sets message for the current test case.

If this keyword is used outside a test teardown, the message is overridden by possible failure message. If this is used in teardown, possible earlier failure message is overridden. Failures in teardown are always shown in addition to this message.

If the optional `append` argument is given any value considered `true` in Python, for example, any non-empty string, the given message is added after the possible earlier message by joining the messages with a space.

In teardown the current test message is available as a built-in variable `$[TEST MESSAGE]`. This keyword can not be used in suite setup or suite teardown.

Examples: | Set Test Message | My message | | Set Test Message | is continued. | append=yes | | Should Be Equal | ${TEST MESSAGE} | My message is continued. |

New in Robot Framework 2.5. Support for `append` was added in 2.7.7.

**set_test_variable** (*name, *values*)

Makes a variable available everywhere within the scope of the current test.

Variables set with this keyword are available everywhere within the scope of the currently executed test case. For example, if you set a variable in a user keyword, it is available both in the test case level and also in all other user keywords used in the current test. Other test cases will not see variables set with this keyword.

See **Set Suite Variable** for more information and examples.

**set_variable** (*values*)

Returns the given values which can then be assigned to a variables.
This keyword is mainly used for setting scalar variables. Additionally it can be used for converting a scalar variable containing a list to a list variable or to multiple scalar variables. It is recommended to use Create List when creating new lists.

Examples: | ${hi} = | Set Variable | Hello, world! | | ${hi2} = | Set Variable | I said: | | ${list} = | Set Variable | @{list} = | | @{list} = | Set Variable | ${list with some items} | | ${item1} | ${item2} = | Set Variable | ${list with 2 items} |

Variables created with this keyword are available only in the scope where they are created. See Set Global Variable, Set Test Variable and Set Suite Variable for information on how to set variables so that they are available also in a larger scope.

**set_variable_if** *(condition, *values)*

Sets variable based on the given condition.

The basic usage is giving a condition and two values. The given condition is first evaluated the same way as with the Should Be True keyword. If the condition is true, then the first value is returned, and otherwise the second value is returned. The second value can also be omitted, in which case it has a default value None. This usage is illustrated in the examples below, where $rc$ is assumed to be zero.

${var1} = | Set Variable If | $rc == 0 | zero | nonzero |
${var2} = | Set Variable If | $rc > 0 | value1 | value2 |
${var3} = | Set Variable If | $rc > 0 | whatever |

=>$rc$ = 'zero' | $var2 = 'value2' | $var3 = None

It is also possible to have ‘Else If’ support by replacing the second value with another condition, and having two new values after it. If the first condition is not true, the second is evaluated and one of the values after it is returned based on its truth value. This can be continued by adding more conditions without a limit.

${var} = | Set Variable If | $rc == 0 | zero |
... | $rc > 0 | greater than zero | less then zero |
... | $rc == 1 | one |
... | $rc == 2 | two |
... | $rc > 2 | greater than two |
... | $rc < 0 | less than zero |

Use Get Variable Value if you need to set variables dynamically based on whether a variable exist or not.

**should_be_empty** *(item, msg=None)*

Verifies that the given item is empty.

The length of the item is got using the Get Length keyword. The default error message can be overridden with the msg argument.

**should_be_equal** *(first, second, msg=None, values=True)*

Fails if the given objects are unequal.

•If msg is not given, the error message is ‘first != second’.

40 Chapter 4. All packages
- If `msg` is given and `values` is either Boolean False or the string ‘False’ or ‘No Values’, the error message is simply `msg`.
- Otherwise the error message is 'msg: first != second'.

**should_be_equal_as_integers** (first, second, msg=None, values=True, base=None)

Fails if objects are unequal after converting them to integers.

See *Convert To Integer* for information how to convert integers from other bases than 10 using `base` argument or `0b/0o/0x` prefixes.

See *Should Be Equal* for an explanation on how to override the default error message with `msg` and `values`.

Examples:

| Should Be Equal As Integers | 42 | ${42} | Error message |
| Should Be Equal As Integers | ABCD | abcd | base=16 |

**should_be_equal_as_numbers** (first, second, msg=None, values=True, precision=6)

Fails if objects are unequal after converting them to real numbers.

The conversion is done with *Convert To Number* keyword using the given `precision`. The support for giving precision was added in Robot Framework 2.6, in earlier versions it was hard-coded to 6.

Examples:

| Should Be Equal As Numbers | ${x} | 1.1 | # Passes if ${x} is 1.1 |
| Should Be Equal As Numbers | 1.123 | 1.1 | precision=1 |

As discussed in the documentation of *Convert To Number*, machines generally cannot store floating point numbers accurately. Because of this limitation, comparing floats for equality is problematic and a correct approach to use depends on the context. This keyword uses a very naive approach of rounding the numbers before comparing them, which is both prone to rounding errors and does not work very well if numbers are really big or small. For more information about comparing floats, and ideas on how to implement your own context specific comparison algorithm, see this great article: [http://www.cygnus-software.com/papers/comparingfloats/comparingfloats.htm](http://www.cygnus-software.com/papers/comparingfloats/comparingfloats.htm)

See *Should Not Be Equal As Numbers* for a negative version of this keyword and *Should Be Equal* for an explanation on how to override the default error message with `msg` and `values`.

**should_be_equal_as_strings** (first, second, msg=None, values=True)

Fails if objects are unequal after converting them to strings.

See *Should Be Equal* for an explanation on how to override the default error message with `msg` and `values`.

**should_be_true** (condition, msg=None)

Fails if the given condition is not true.

If `condition` is a string (e.g. ‘${rc} < 10’), it is evaluated as a Python expression using the built-in ‘eval’ function and the keyword status is decided based on the result. If a non-string item is given, the status is got directly from its truth value as explained at [http://docs.python.org/lib/truth.html](http://docs.python.org/lib/truth.html).

The default error message (‘<condition> should be true’) is not very informative, but it can be overridden with the `msg` argument.

Examples:

| Should Be True | ${rc} < 10 | Should Be True | ‘${status}’ == ‘PASS’ |
| Should Be True | ${number} | # Passes if ${number} is not zero |

**should_contain** (item1, item2, msg=None, values=True)

Fails if `item1` does not contain `item2` one or more times.

Works with strings, lists, and anything that supports Python’s ‘in’ keyword. See *Should Be Equal* for an explanation on how to override the default error message with `msg` and `values`.

Examples:

| Should Contain | ${output} | PASS |
| Should Contain | ${some_list} | value |
should_contain_x_times(item1, item2, count, msg=None)
Fails if item1 does not contain item2 count times.

Works with strings, lists and all objects that Get Count works with. The default error message can be overridden with msg and the actual count is always logged.

Examples:
1 Should Contain X Times | ${output} | hello | 2 | Should Contain X Times | ${some list} | value | 1 |

should_end_with(str1, str2, msg=None, values=True)
Fails if the string str1 does not end with the string str2.

See Should Be Equal for an explanation on how to override the default error message with msg and values.

should_match(string, pattern, msg=None, values=True)
Fails unless the given string matches the given pattern.

Pattern matching is similar as matching files in a shell, and it is always case-sensitive. In the pattern, ‘*’ matches to anything and ‘?’ matches to any single character.

See Should Be Equal for an explanation on how to override the default error message with msg and values.

should_match_regexp(string, pattern, msg=None, values=True)
Fails if string does not match pattern as a regular expression.

Regular expression check is done using the Python ‘re’ module, which has a pattern syntax derived from Perl, and thus also very similar to the one in Java. See the following documents for more details about regular expressions in general and Python implementation in particular.

http://docs.python.org/lib/module-re.html
http://www.amk.ca/python/howto/regex/

Things to note about the regexp syntax in Robot Framework test data:

1) Backslash is an escape character in the test data, and possible backslashes in the pattern must thus be escaped with another backslash (e.g. '\d\w+').

2) Strings that may contain special characters, but should be handled as literal strings, can be escaped with the Regexp Escape keyword.

3) The given pattern does not need to match the whole string. For example, the pattern ‘ello’ matches the string ‘Hello world!’’. If a full match is needed, the ‘^’ and ‘$’ characters can be used to denote the beginning and end of the string, respectively. For example, ‘^ello$’ only matches the exact string ‘ello’.

4) Possible flags altering how the expression is parsed (e.g. re.IGNORECASE, re.MULTILINE) can be set by prefixing the pattern with the ‘(?iLmsux)’ group (e.g. ‘(?im)pattern’). The available flags are ‘IGNORECASE’: ‘i’, ‘MULTILINE’: ‘m’, ‘DOTALL’: ‘s’, ‘VERBOSE’: ‘x’, ‘_UNICODE’: ‘u’, and ‘_LOCALE’: ‘L’.

If this keyword passes, it returns the portion of the string that matched the pattern. Additionally, the possible captured groups are returned.

See the Should Be Equal keyword for an explanation on how to override the default error message with the msg and values arguments.

Examples:
1 Should Match Regexp | $\{output\} | \d{6} | # Output contains six numbers | Should Match Regexp | $\{output\} | ^\d{6}$ | # Six numbers and nothing more | Should Match Regexp | Foo: 42 | (?i)foo: $\+|match|$\{group1\}|$\{group2\} = $\{ret\} | Should Match Regexp | Bar: 43 | (FooBar): ($\+| => |$\{ret\} = ‘Foo: 42’ |$\{match\} = ‘Bar: 43’ |$\{group1\} = ‘Bar’ |$\{group2\} = ‘43’
should_not_be_empty (item, msg=None)
Verifies that the given item is not empty.

The length of the item is got using the Get Length keyword. The default error message can be overridden with the msg argument.

should_not_be_equal (first, second, msg=None, values=True)
Fails if the given objects are equal.

See Should Be Equal for an explanation on how to override the default error message with msg and values.

should_not_be_equal_as_integers (first, second, msg=None, values=True, base=None)
Fails if objects are equal after converting them to integers.

See Convert To Integer for information how to convert integers from other bases than 10 using base argument or 0b/0o/0x prefixes.

See Should Be Equal for an explanation on how to override the default error message with msg and values.

See Should Be Equal As Integers for some usage examples.

should_not_be_equal_as_numbers (first, second, msg=None, values=True, precision=6)
Fails if objects are equal after converting them to real numbers.

The conversion is done with Convert To Number keyword using the given precision. The support for giving precision was added in Robot Framework 2.6, in earlier versions it was hard-coded to 6.

See Should Be Equal As Numbers for examples on how to use precision and why it does not always work as expected. See also Should Be Equal for an explanation on how to override the default error message with msg and values.

should_not_be_equal_as_strings (first, second, msg=None, values=True)
Fails if objects are equal after converting them to strings.

See Should Be Equal for an explanation on how to override the default error message with msg and values.

should_not_be_true (condition, msg=None)
Fails if the given condition is true.

See Should Be True for details about how condition is evaluated and how msg can be used to override the default error message.

should_not_contain (item1, item2, msg=None, values=True)
Fails if item1 contains item2 one or more times.

Works with strings, lists, and anything that supports Python’s ‘in’ keyword. See Should Be Equal for an explanation on how to override the default error message with msg and values.

Examples: | Should Not Contain | ${output} | FAILED | | Should Not Contain | ${some_list} | value |

should_not_end_with (str1, str2, msg=None, values=True)
Fails if the string str1 ends with the string str2.

See Should Be Equal for an explanation on how to override the default error message with msg and values.

should_not_match (string, pattern, msg=None, values=True)
Fails if the given string matches the given pattern.

Pattern matching is similar as matching files in a shell, and it is always case-sensitive. In the pattern ‘*’ matches to anything and ‘?’ matches to any single character.

See Should Be Equal for an explanation on how to override the default error message with msg and values.

should_not_match_regexp (string, pattern, msg=None, values=True)
Fails if string matches pattern as a regular expression.
See Should Match Regexp for more information about arguments.

**should_not_start_with** *(str1, str2, msg=None, values=True)*  
Fails if the string `str1` starts with the string `str2`.

See Should Be Equal for an explanation on how to override the default error message with `msg` and `values`.

**should_start_with** *(str1, str2, msg=None, values=True)*  
Fails if the string `str1` does not start with the string `str2`.

See Should Be Equal for an explanation on how to override the default error message with `msg` and `values`.

**sleep** *(time, reason=None)*  
Pauses the test executed for the given time.

`time` may be either a number or a time string. Time strings are in a format such as `‘1 day 2 hours 3 minutes 4 seconds 5milliseconds’` or `‘1d 2h 3m 4s 5ms’`, and they are fully explained in an appendix of Robot Framework User Guide. Optional `reason` can be used to explain why sleeping is necessary. Both the time slept and the reason are logged.

Examples: | Sleep | 42 | | Sleep | 1.5 | | Sleep | 2 minutes 10 seconds | | Sleep | 10s | Wait for a reply |

**variable_should_exist** *(name, msg=None)*  
Fails unless the given variable exists within the current scope.

The name of the variable can be given either as a normal variable name (e.g. `${NAME}`) or in escaped format (e.g. `\${NAME}`). Notice that the former has some limitations explained in Set Suite Variable.

The default error message can be overridden with the `msg` argument.

See also Variable Should Not Exist and Keyword Should Exist.

**variable_should_not_exist** *(name, msg=None)*  
Fails if the given variable exists within the current scope.

The name of the variable can be given either as a normal variable name (e.g. `${NAME}`) or in escaped format (e.g. `\${NAME}`). Notice that the former has some limitations explained in Set Suite Variable.

The default error message can be overridden with the `msg` argument.

See also Variable Should Exist and Keyword Should Exist.

**wait_until_keyword_succeeds** *(timeout, retry_interval, name, *args)*  
Waits until the specified keyword succeeds or the given timeout expires.

`name` and `args` define the keyword that is executed similarly as with Run Keyword. If the specified keyword does not succeed within `timeout`, this keyword fails. `retry_interval` is the time to wait before trying to run the keyword again after the previous run has failed.

Both `timeout` and `retry_interval` must be given in Robot Framework’s time format (e.g. ‘1 minute’, ‘2 min 3 s’, ‘4.5’).

Errors caused by invalid syntax, test or keyword timeouts, or fatal exceptions are not caught by this keyword.

Example: | Wait Until Keyword Succeeds | 2 min | 5 sec | My keyword | arg1 | arg2 |

Running the same keyword multiple times inside this keyword can create lots of output and considerably increase the size of the generated output files. Starting from Robot Framework 2.7, it is possible to remove unnecessary keywords from the outputs using --RemoveKeywords WUKS command line option.

**robot.libraries.BuiltIn.register_run_keyword** *(library, keyword, args_to_process=None)*  
Registers ‘run keyword’ so that its arguments can be handled correctly.

1. Why is this method needed
Keywords running other keywords internally (normally using Run Keyword or some variants of it in BuiltIn) must have the arguments meant to the internally executed keyword handled specially to prevent processing them twice. This is done ONLY for keywords registered using this method.

If the register keyword has same name as any keyword from Robot Framework standard libraries, it can be used without getting warnings. Normally there is a warning in such cases unless the keyword is used in long format (e.g. MyLib.Keyword).

Starting from Robot Framework 2.5.2, keywords executed by registered run keywords can be tested in dry-run mode they have ‘name’ argument which takes the name of the executed keyword.

2. How to use this method

library is the name of the library where the registered keyword is implemented.

keyword can be either a function or method implementing the keyword, or name of the implemented keyword as a string.

args_to_process is needed when keyword is given as a string, and it defines how many of the arguments to the registered keyword must be processed normally. When keyword is a method or function, this information is got directly from it so that varargs (those specified with syntax ‘*args’) are not processed but others are.

3. Examples

from robot.libraries.BuiltIn import BuiltIn, register_run_keyword
def my_run_keyword(name, *args):
    # do something return BuiltIn().run_keyword(name, *args)
# Either one of these works register_run_keyword(__name__, my_run_keyword) register_run_keyword(__name__, ‘My Run Keyword’, 1)
from robot.libraries.BuiltIn import BuiltIn, register_run_keyword
class MyLibrary:
    def my_run_keyword_if(self, expression, name, *args):
        # do something return
        BuiltIn().run_keyword_if(expression, name, *args)
# Either one of these works register_run_keyword(‘MyLibrary’, MyLibrary.my_run_keyword_if) register_run_keyword(‘MyLibrary’, ‘my_run_keyword_if’, 2)

Collections Module

class robot.libraries.Collections.Collections
A test library providing keywords for handling lists and dictionaries.

Collections is Robot Framework’s standard library that provides a set of keywords for handling Python lists and dictionaries. This library has keywords, for example, for modifying and getting values from lists and dictionaries (e.g. Append To List, Get From Dictionary) and for verifying their contents (e.g. Lists Should Be Equal, Dictionary Should Contain Value).

Following keywords from the BuiltIn library can also be used with lists and dictionaries: | Keyword Name | Applicable With |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Create List</td>
<td>lists</td>
</tr>
<tr>
<td>Get Length</td>
<td>both</td>
</tr>
<tr>
<td>Length Should Be</td>
<td>both</td>
</tr>
<tr>
<td>Should Be Empty</td>
<td>both</td>
</tr>
<tr>
<td>Should Not Be Empty</td>
<td>both</td>
</tr>
<tr>
<td>Should Contain</td>
<td>lists</td>
</tr>
<tr>
<td>Should Not Contain</td>
<td>lists</td>
</tr>
<tr>
<td>Should Contain X Times</td>
<td>lists</td>
</tr>
<tr>
<td>Should Not Contain X Times</td>
<td>lists</td>
</tr>
<tr>
<td>Get Count</td>
<td>lists</td>
</tr>
</tbody>
</table>

All list keywords expect a scalar variable (e.g. ${list}) as an argument. It is, however, possible to use list variables (e.g. @[list]) as scalars simply by replacing ‘@’ with ‘$’.

List keywords that do not alter the given list can also be used with tuples, and to some extend also with other iterables. Convert To List can be used to convert tuples and other iterables to lists.

4.1. robot Package
List related keywords use variables in format ${Lx}$ in their examples, which means a list with as many alphabetic characters as specified by ‘x’. For example ${L1}$ means [‘a’] and ${L3}$ means [‘a’, ‘b’, ‘c’].

Dictionary keywords use similar ${Dx}$ variables. For example ${D1}$ means {‘a’: 1} and ${D3}$ means {‘a’: 1, ‘b’: 2, ‘c’: 3}.

ROBOT_LIBRARY_SCOPE = ‘GLOBAL’

ROBOT_LIBRARY_VERSION = ‘2.7.7’

append_to_list (list_, *values)
   Adds values to the end of list.
   Example: | Append To List | ${L1} | xxx | | | | Append To List | ${L2} | x | y | z | => - ${L1} = [‘a’, ‘xxx’] - ${L2} = [‘a’, ‘b’, ‘x’, ‘y’, ‘z’]

combine_lists (*lists)
   Combines the given lists together and returns the result.
   The given lists are not altered by this keyword.
   Example: | Combine Lists | ${L1} | ${L2} | ${L1} | => - ${x} = [‘a’, ‘a’, ‘b’, ‘c’] - ${y} = [‘a’, ‘a’, ‘b’, ‘c’] - ${L1} and ${L2} are not changed.

count_values_in_list (list_, value, start=0, end=None)
   Returns the number of occurrences of the given value in list.
   The search can be narrowed to the selected sublist by the start and end indexes having the same semantics as in the Get Slice From List keyword. The given list is never altered by this keyword.
   Example: | Count Values In List | ${L3} | b | => - ${x} = 1 - ${L3} is not changed

copy_dictionary (dictionary)
   Returns a copy of the given dictionary.
   The given dictionary is never altered by this keyword.

copy_list (list_)
   Returns a copy of the given list.
   The given list is never altered by this keyword.

count_values_in_list (list_, value, start=0, end=None)
   Returns the number of occurrences of the given value in list.
   The search can be narrowed to the selected sublist by the start and end indexes having the same semantics as in the Get Slice From List keyword. The given list is never altered by this keyword.
   Example: | Count Values In List | ${L3} | b | => - ${x} = 1 - ${L3} is not changed

copy_dictionary (dictionary)
   Returns a copy of the given dictionary.
   The given dictionary is never altered by this keyword.

copy_list (list_)
   Returns a copy of the given list.
   The given list is never altered by this keyword.

dictionaries_should_be_equal (dict1, dict2, msg=None, values=True)
   Fails if the given dictionaries are not equal.
   First the equality of dictionaries’ keys is checked and after that all the key value pairs. If there are differences between the values, those are listed in the error message.
   See Lists Should Be Equal for an explanation of msg. The given dictionaries are never altered by this keyword.
**dictionary_should_contain_key** *(dictionary, key, msg=None)*
Fails if key is not found from dictionary.

See *List Should Contain Value* for an explanation of msg.

The given dictionary is never altered by this keyword.

**dictionary_should_contain_sub_dictionary** *(dict1, dict2, msg=None, values=True)*
Fails unless all items in dict2 are found from dict1.

See *Lists Should Be Equal* for an explanation of msg. The given dictionaries are never altered by this keyword.

**dictionary_should_contain_value** *(dictionary, value, msg=None)*
Fails if value is not found from dictionary.

See *List Should Contain Value* for an explanation of msg.

The given dictionary is never altered by this keyword.

**dictionary_should_not_contain_key** *(dictionary, key, msg=None)*
Fails if key is found from dictionary.

See *List Should Contain Value* for an explanation of msg.

The given dictionary is never altered by this keyword.

**dictionary_should_not_contain_value** *(dictionary, value, msg=None)*
Fails if value is found from dictionary.

See *List Should Contain Value* for an explanation of msg.

The given dictionary is never altered by this keyword.

**get_dictionary_items** *(dictionary)*
Returns items of the given dictionary.

Items are returned sorted by keys. The given dictionary is not altered by this keyword.

Example: | ${items} = | Get Dictionary Items | ${D3} | => - ${items} = ['a', 1, 'b', 2, 'c', 3]

**get_dictionary_keys** *(dictionary)*
Returns keys of the given dictionary.

Keys are returned in sorted order. The given dictionary is never altered by this keyword.

Example: | ${keys} = | Get Dictionary Keys | ${D3} | => - ${keys} = ['a', 'b', 'c']

**get_dictionary_values** *(dictionary)*
Returns values of the given dictionary.

Values are returned sorted according to keys. The given dictionary is never altered by this keyword.

Example: | ${values} = | Get Dictionary Values | ${D3} | => - ${values} = [1, 2, 3]

**get_from_dictionary** *(dictionary, key)*
Returns a value from the given dictionary based on the given key.

If the given key cannot be found from the dictionary, this keyword fails.

The given dictionary is never altered by this keyword.

Example: | ${value} = | Get From Dictionary | ${D3} | b | => - ${value} = 2

**get_from_list** *(list, index)*
Returns the value specified with an index from list.

The given list is never altered by this keyword.
Index ‘0’ means the first position, ‘1’ the second, and so on. Similarly, ‘-1’ is the last position, ‘-2’ the second last, and so on. Using an index that does not exist on the list causes an error. The index can be either an integer or a string that can be converted to an integer.

Examples (including Python equivalents in comments):  
| ${x} = | Get From List | ${L5} | 0 | # L5[0] | 1 | Get From List | ${L5} | -2 | # L5[-2] | => - $\{x\} = \{'a': $\{'y': - 2 | # L5 is not changed

get_index_from_list (list\_, value, start=0, end=None)  
Returns the index of the first occurrence of the value on the list.

The search can be narrowed to the selected sublist by the start and end indexes having the same semantics as in the Get Slice From List keyword. In case the value is not found, -1 is returned. The given list is never altered by this keyword.

Example:  | Insert Into List | ${L1} | 0 | xxx | => - ${L1} = ['xxx', 'a']

get_slice_from_list (list\_, start=0, end=None)  
Returns a slice of the given list between start and end indexes.

The given list is never altered by this keyword.

If both start and end are given, a sublist containing values from start to end is returned. This is the same as list[start:end] in Python. To get all items from the beginning, use 0 as the start value, and to get all items until the end, use ‘None’ as the end value. ‘None’ is also a default value, so in this case, it is enough to give only start. If only end is given, start gets the value 0.

Using start or end not found on the list is the same as using the largest (or smallest) available index.

Examples (incl. Python equivalents in comments):  | Insert Into List | ${L2} | ${-1} | xxx | => - ${L2} = ['a', 'xxx', 'b']

insert_into_list (list\_, index, value)  
Inserts value into list to the position specified with index.

Index ‘0’ adds the value into the first position, ‘1’ to the second, and so on. Inserting from right works with negative indices so that ‘-1’ is the second last position, ‘-2’ third last, and so on. Use Append To List to add items to the end of the list.

If the absolute value of the index is greater than the length of the list, the value is added at the end (positive index) or the beginning (negative index). An index can be given either as an integer or a string that can be converted to an integer.

Example:  | Insert Into List | ${L1} | 0 | xxx | => - ${L1} = ['xxx', 'a']

keep_in_dictionary (dictionary, *keys)  
Keeps the given keys in the dictionary and removes all other.

If the given key cannot be found from the dictionary, it is ignored.

Example:  | Keep In Dictionary | ${D5} | b | x | d | => - ${D5} = {'b': 2, 'd': 4}

list_should_contain_sub_list (list1, list2, msg=None, values=True)  
Fails if not all of the elements in list2 are found in list1.

The order of values and the number of values are not taken into account.

See the use of msg and values from the Lists Should Be Equal keyword.

list_should_contain_value (list\_, value, msg=None)  
Fails if the value is not found from list.
If *msg* is not given, the default error message “[ a b l c ] does not contain the value ‘x’” is shown in case of a failure. Otherwise, the given *msg* is used in case of a failure.

**list_should_not_contain_duplicates** *(list, msg=None)*  
Fails if any element in the *list* is found from it more than once.  
The default error message lists all the elements that were found from the *list* multiple times, but it can be overridden by giving a custom *msg*. All multiple times found items and their counts are also logged.

This keyword works with all iterables that can be converted to a list. The original iterable is never altered.

**list_should_not_contain_value** *(list, value, msg=None)*  
Fails if the *value* is not found from *list*.

See List Should Contain Value for an explanation of *msg*.

**lists_should_be_equal** *(list1, list2, msg=None, values=True, names=None)*  
Fails if given lists are unequal.

The keyword first verifies that the lists have equal lengths, and then it checks are all their values equal. Possible differences between the values are listed in the default error message like *Index 4: ABC != Abc*.

The error message can be configured using *msg* and *values* arguments: - If *msg* is not given, the default error message is used. - If *msg* is given and *values* is either Boolean False or a string ‘False’ or ‘No Values’, the error message is simply *msg*.

•Otherwise the error message is *msg* + ‘new line’ + default.

Optional *names* argument (new in 2.6) can be used for naming the indices shown in the default error message. It can either be a list of names matching the indices in the lists or a dictionary where keys are indices that need to be named. It is not necessary to name all of the indices. When using a dictionary, keys can be either integers or strings that can be converted to integers.

Examples:  
| *names* = | Create List | First Name | Family Name | Email | Lists Should Be Equal |  
|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|
| | First Name | Family Name | Email | Lists Should Be Equal |  
| = | First Name | Family Name | Email | Lists Should Be Equal |  
| $\{people1\} | $\{people2\} | names=$\{names\} | $\{names\} = | Create Dictionary | 1 | First Name | 2 | Email | Lists Should Be Equal |  
| $\{people1\} | $\{people2\} | names=$\{names\} |}

If the items in index 2 would differ in the above examples, the error message would contain a row like *Index 2 (email): name@foo.com != name@bar.com*.

**log_dictionary** *(dictionary, level='INFO')*  
Logs the size and contents of the *dictionary* using given *level*.

Valid levels are TRACE, DEBUG, INFO (default), and WARN.  
If you only want to log the size, use keyword *Get Length* from the BuiltIn library.

**log_list** *(list, level='INFO')*  
Logs the length and contents of the *list* using given *level*.

Valid levels are TRACE, DEBUG, INFO (default), and WARN.  
If you only want to the length, use keyword *Get Length* from the BuiltIn library.

**remove_duplicates** *(list,)*  
Returns a list without duplicates based on the given *list*.

Creates and returns a new list that contains all items in the given list so that one item can appear only once. Order of the items in the new list is the same as in the original except for missing duplicates. Number of the removed duplicates is logged.

New in Robot Framework 2.7.5.
remove_from_dictionary \( \text{(dictionary, } * \text{keys}) \)

Removes the given keys from the dictionary.

If the given key cannot be found from the dictionary, it is ignored.

Example: l Remove From Dictionary l ${D3} | b | x | y | => - ${D3} = \{ \text{‘a’: 1, ‘c’: 3} \}

remove_from_list \( \text{(list, index)} \)

Removes and returns the value specified with an index from list.

Index ‘0’ means the first position, ‘1’ the second and so on. Similarly, ‘-1’ is the last position, ‘-2’ the second last, and so on. Using an index that does not exist on the list causes an error. The index can be either an integer or a string that can be converted to an integer.

Example: l $\{x\} = l Remove From List l $\{L2\} | 0 | => - $\{x\} = \text{‘a’} - $\{L2\} = \{'b'\}

remove_values_from_list \( \text{(list, *values)} \)

Removes all occurrences of given values from list.

It is not an error if a value does not exist in the list at all.

Example: l Remove Values From List l $\{L4\} | a | c | e | f | => - $\{L4\} = \{'b', 'd'\}

reverse_list \( \text{(list)} \)

Reverses the given list in place.

Note that the given list is changed and nothing is returned. Use Copy List first, if you need to keep also the original order.

Reverse List l $\{L3\} |

=> - $\{L3\} = \{'c', 'b', 'a'\}

set_list_value \( \text{(list, index, value)} \)

Sets the value of list specified by index to the given value.

Index ‘0’ means the first position, ‘1’ the second and so on. Similarly, ‘-1’ is the last position, ‘-2’ second last, and so on. Using an index that does not exist on the list causes an error. The index can be either an integer or a string that can be converted to an integer.

Example: l Set List Value l $\{L3\} | 1 | xxx | l Set List Value l $\{L3\} | -1 | yyy | => - $\{L3\} = \{'a', 'xxx', 'yyy'\}

set_to_dictionary \( \text{(dictionary, } * \text{key_value_pairs)} \)

Adds the given key_value_pairs to the dictionary.

Example: l Set To Dictionary l $\{D1\} | key | value | => - $\{D1\} = \{\text{‘a’: 1, ‘key’: ‘value’}\}

sort_list \( \text{(list)} \)

Sorts the given list in place.

The strings are sorted alphabetically and the numbers numerically.

Note that the given list is changed and nothing is returned. Use Copy List first, if you need to keep also the original order.

$\{L\} = \{2,1,’a’,’c’,’b’\} l Sort List l $\{L\} | => - $\{L\} = \{1, 2, ‘a’, ‘b’, ‘c’\}
**DeprecatedBuiltIn Module**

```python
class robot.libraries.DeprecatedBuiltIn.DeprecatedBuiltIn

    ROBOT_LIBRARY_SCOPE = 'GLOBAL'

classmethod integer(item, base=None)

    Converts the given item to an integer number.

    If the given item is a string, it is by default expected to be an integer in base 10. Starting from Robot Framework 2.6 there are two ways to convert from other bases:

    1. Give base explicitly to the keyword as `base` argument.

    2) Prefix the given string with the base so that `0b` means binary (base 2), `0o` means octal (base 8), and `0x` means hex (base 16). The prefix is considered only when `base` argument is not given and may itself be prefixed with a plus or minus sign.

    The syntax is case-insensitive and possible spaces are ignored.

    Examples:
    | $result = Convert To Integer 100 | # Result is 100 |
    | $result = Convert To Integer FF AA 16 | # Result is 65450 |
    | $result = Convert To Integer 100 8 | # Result is 64 |
    | $result = Convert To Integer -100 2 | # Result is -4 |
    | $result = Convert To Integer 0b100 | # Result is 4 |
    | $result = Convert To Integer -0x100 | # Result is -256 |

    See also `Convert To Number`, `Convert To Binary`, `Convert To Octal` and `Convert To Hex`.

classmethod float(item, precision=None)

    Converts the given item to a floating point number.

    If the optional `precision` is positive or zero, the returned number is rounded to that number of decimal digits. Negative precision means that the number is rounded to the closest multiple of 10 to the power of the absolute precision. The support for precision was added in Robot Framework 2.6.

    Examples:
    | $result = Convert To Number 42.512 | # Result is 42.512 |
    | $result = Convert To Number 42.512 1 | # Result is 42.5 |
    | $result = Convert To Number 42.512 0 | # Result is 43.0 |
    | $result = Convert To Number 42.512 -1 | # Result is 40.0 |

    Notice that machines generally cannot store floating point numbers accurately. This may cause surprises with these numbers in general and also when they are rounded. For more information see, for example, this floating point arithmetic tutorial: [http://docs.python.org/tutorial/floatingpoint.html](http://docs.python.org/tutorial/floatingpoint.html)

    If you need an integer number, use `Convert To Integer` instead.

classmethod string(item)

    Converts the given item to a Unicode string.

    Uses '__unicode__' or '__str__' method with Python objects and 'toString' with Java objects.

    Use `Encode String To Bytes` and `Decode Bytes To String` keywords in `String` library if you need to convert between Unicode and byte strings.

classmethod boolean(item)

    Converts the given item to Boolean true or false.

    Handles strings 'True' and 'False' (case-insensitive) as expected, otherwise returns item's truth value using Python's 'bool' method. For more information about truth values, see [http://docs.python.org/lib/bool.html](http://docs.python.org/lib/bool.html).

classmethod list(*items)

    Returns a list containing given items.

    The returned list can be assigned both to `${scalar}` and `@{list}` variables. The earlier can be used e.g. with Java keywords expecting an array as an argument.
```
Examples:  l @|list} = l Create List l a l b l c l |$scalar} = l Create List l a l b l c l |${ints} = l Create List |${1} |${2} |${3} |

classmethod equal (first, second, msg=None, values=True)  
Fails if the given objects are unequal.
  • If msg is not given, the error message is ‘first != second’.
  • If msg is given and values is either Boolean False or the string ‘False’ or ‘No Values’, the error message is simply msg.
  • Otherwise the error message is ‘msg: first != second’.

classmethod equals (first, second, msg=None, values=True)  
Fails if the given objects are unequal.
  • If msg is not given, the error message is ‘first != second’.
  • If msg is given and values is either Boolean False or the string ‘False’ or ‘No Values’, the error message is simply msg.
  • Otherwise the error message is ‘msg: first != second’.

classmethod fail_unless_equal (first, second, msg=None, values=True)  
Fails if the given objects are unequal.
  • If msg is not given, the error message is ‘first != second’.
  • If msg is given and values is either Boolean False or the string ‘False’ or ‘No Values’, the error message is simply msg.
  • Otherwise the error message is ‘msg: first != second’.

classmethod not_equal (first, second, msg=None, values=True)  
Fails if the given objects are equal.

See Should Be Equal for an explanation on how to override the default error message with msg and values.

classmethod not_equals (first, second, msg=None, values=True)  
Fails if the given objects are equal.

See Should Be Equal for an explanation on how to override the default error message with msg and values.

classmethod fail_if_equal (first, second, msg=None, values=True)  
Fails if the given objects are equal.

See Should Be Equal for an explanation on how to override the default error message with msg and values.

classmethod is_true (condition, msg=None)  
Fails if the given condition is not true.

If condition is a string (e.g. ‘${rc} < 10’), it is evaluated as a Python expression using the built-in ‘eval’ function and the keyword status is decided based on the result. If a non-string item is given, the status is got directly from its truth value as explained at http://docs.python.org/lib/truth.html.

The default error message (‘<condition> should be true’) is not very informative, but it can be overridden with the msg argument.

Examples:  l Should Be True l ${rc} < 10 l Should Be True l ‘${status}’ == ‘PASS’ l # Strings must be quoted l Should Be True l ${number} l # Passes if ${number} is not zero l Should Be True l ${list} l # Passes if ${list} is not empty l

classmethod fail_unless (condition, msg=None)  
Fails if the given condition is not true.
If `condition` is a string (e.g. `{$rc} < 10`), it is evaluated as a Python expression using the built-in ‘eval’ function and the keyword status is decided based on the result. If a non-string item is given, the status is got directly from its truth value as explained at http://docs.python.org/lib/truth.html.

The default error message (‘<condition> should be true’) is not very informative, but it can be overridden with the `msg` argument.

Examples: | Should Be True | `{$rc}` < 10 | | Should Be True | `{$status}` == ‘PASS’ | # Strings must be quoted | Should Be True | `{$number}` | # Passes if `{$number}` is not zero | Should Be True | `{$list}` | # Passes if `{$list}` is not empty |

```python
classmethod is_false(condition, msg=None)
Fails if the given condition is true.
See Should Be True for details about how `condition` is evaluated and how `msg` can be used to override the default error message.
```

```python
classmethod fail_if(condition, msg=None)
Fails if the given condition is true.
See Should Be True for details about how `condition` is evaluated and how `msg` can be used to override the default error message.
```

```python
classmethod fail_if_ints_equal(first, second, msg=None, values=True, base=None)
Fails if objects are equal after converting them to integers.
See Convert To Integer for information how to convert integers from other bases than 10 using `base` argument or `0b/0o/0x` prefixes.
See Should Be Equal for an explanation on how to override the default error message with `msg` and `values`.
See Should Be Equal As Integers for some usage examples.
```

```python
classmethod ints_not_equal(first, second, msg=None, values=True, base=None)
Fails if objects are equal after converting them to integers.
See Convert To Integer for information how to convert integers from other bases than 10 using `base` argument or `0b/0o/0x` prefixes.
See Should Be Equal for an explanation on how to override the default error message with `msg` and `values`.
See Should Be Equal As Integers for some usage examples.
```

```python
classmethod ints_equal(first, second, msg=None, values=True, base=None)
Fails if objects are unequal after converting them to integers.
See Convert To Integer for information how to convert integers from other bases than 10 using `base` argument or `0b/0o/0x` prefixes.
See Should Be Equal for an explanation on how to override the default error message with `msg` and `values`.
Examples: | Should Be Equal As Integers | 42 | `{$42}` | Error message | Should Be Equal As Integers | ABCD | abcd | base=16 | Should Be Equal As Integers | 0b1011 | 11 |
```

```python
classmethod fail_unless_ints_equal(first, second, msg=None, values=True, base=None)
Fails if objects are unequal after converting them to integers.
See Convert To Integer for information how to convert integers from other bases than 10 using `base` argument or `0b/0o/0x` prefixes.
See Should Be Equal for an explanation on how to override the default error message with `msg` and `values`.
Examples: | Should Be Equal As Integers | 42 | `{$42}` | Error message | Should Be Equal As Integers | ABCD | abcd | base=16 | Should Be Equal As Integers | 0b1011 | 11 |
```
classmethod floats_not_equal (first, second, msg=None, values=True, precision=6)
Fails if objects are equal after converting them to real numbers.

The conversion is done with Convert To Number keyword using the given precision. The support for giving precision was added in Robot Framework 2.6, in earlier versions it was hard-coded to 6.

See Should Be Equal As Numbers for examples on how to use precision and why it does not always work as expected. See also Should Be Equal for an explanation on how to override the default error message with msg and values.

classmethod fail_if_floats_equal (first, second, msg=None, values=True, precision=6)
Fails if objects are equal after converting them to real numbers.

The conversion is done with Convert To Number keyword using the given precision. The support for giving precision was added in Robot Framework 2.6, in earlier versions it was hard-coded to 6.

See Should Be Equal As Numbers for examples on how to use precision and why it does not always work as expected. See also Should Be Equal for an explanation on how to override the default error message with msg and values.

classmethod floats_equal (first, second, msg=None, values=True, precision=6)
Fails if objects are unequal after converting them to real numbers.

The conversion is done with Convert To Number keyword using the given precision. The support for giving precision was added in Robot Framework 2.6, in earlier versions it was hard-coded to 6.

Examples: | Should Be Equal As Numbers | ${x} | 1.1 | | # Passes if ${x} is 1.1 | | Should Be Equal As Numbers | 1.123 | 1.1 | precision=1 | # Passes | | Should Be Equal As Numbers | 1.123 | 1.4 | precision=0 | # Passes | | Should Be Equal As Numbers | 112.3 | 75 | precision=-2 | # Passes |

As discussed in the documentation of Convert To Number, machines generally cannot store floating point numbers accurately. Because of this limitation, comparing floats for equality is problematic and a correct approach to use depends on the context. This keyword uses a very naive approach of rounding the numbers before comparing them, which is both prone to rounding errors and does not work very well if numbers are really big or small. For more information about comparing floats, and ideas on how to implement your own context specific comparison algorithm, see this great article: http://www.cygnus-software.com/papers/comparingfloats/comparingfloats.htm

See Should Not Be Equal As Numbers for a negative version of this keyword and Should Be Equal for an explanation on how to override the default error message with msg and values.

classmethod fail_unless_floats_equal (first, second, msg=None, values=True, precision=6)
Fails if objects are unequal after converting them to real numbers.

The conversion is done with Convert To Number keyword using the given precision. The support for giving precision was added in Robot Framework 2.6, in earlier versions it was hard-coded to 6.

Examples: | Should Be Equal As Numbers | ${x} | 1.1 | | # Passes if ${x} is 1.1 | | Should Be Equal As Numbers | 1.123 | 1.1 | precision=1 | # Passes | | Should Be Equal As Numbers | 1.123 | 1.4 | precision=0 | # Passes | | Should Be Equal As Numbers | 112.3 | 75 | precision=-2 | # Passes |

As discussed in the documentation of Convert To Number, machines generally cannot store floating point numbers accurately. Because of this limitation, comparing floats for equality is problematic and a correct approach to use depends on the context. This keyword uses a very naive approach of rounding the numbers before comparing them, which is both prone to rounding errors and does not work very well if numbers are really big or small. For more information about comparing floats, and ideas on how to implement your own context specific comparison algorithm, see this great article: http://www.cygnus-software.com/papers/comparingfloats/comparingfloats.htm

See Should Not Be Equal As Numbers for a negative version of this keyword and Should Be Equal for an explanation on how to override the default error message with msg and values.
classmethod **does_not_start** *(str1, str2, msg=None, values=True)*
Fails if the string `str1` starts with the string `str2`.

See `Should Be Equal` for an explanation on how to override the default error message with `msg` and `values`.

classmethod **fail_if_starts** *(str1, str2, msg=None, values=True)*
Fails if the string `str1` starts with the string `str2`.

See `Should Be Equal` for an explanation on how to override the default error message with `msg` and `values`.

classmethod **starts** *(str1, str2, msg=None, values=True)*
Fails if the string `str1` does not start with the string `str2`.

See `Should Be Equal` for an explanation on how to override the default error message with `msg` and `values`.

classmethod **fail_unless_starts** *(str1, str2, msg=None, values=True)*
Fails if the string `str1` does not start with the string `str2`.

See `Should Be Equal` for an explanation on how to override the default error message with `msg` and `values`.

classmethod **does_not_end** *(str1, str2, msg=None, values=True)*
Fails if the string `str1` ends with the string `str2`.

See `Should Be Equal` for an explanation on how to override the default error message with `msg` and `values`.

classmethod **fail_if_ends** *(str1, str2, msg=None, values=True)*
Fails if the string `str1` ends with the string `str2`.

See `Should Be Equal` for an explanation on how to override the default error message with `msg` and `values`.

classmethod **ends** *(str1, str2, msg=None, values=True)*
Fails if the string `str1` does not end with the string `str2`.

See `Should Be Equal` for an explanation on how to override the default error message with `msg` and `values`.

classmethod **fail_unless_ends** *(str1, str2, msg=None, values=True)*
Fails if the string `str1` does not end with the string `str2`.

See `Should Be Equal` for an explanation on how to override the default error message with `msg` and `values`.

classmethod **does_not_contain** *(item1, item2, msg=None, values=True)*
Fails if `item1` contains `item2` one or more times.

Works with strings, lists, and anything that supports Python’s ‘in’ keyword. See `Should Be Equal` for an explanation on how to override the default error message with `msg` and `values`.

Examples: | Should Not Contain | ${output} | FAILED | | Should Not Contain | ${some_list} | value |

classmethod **fail_if_contains** *(item1, item2, msg=None, values=True)*
Fails if `item1` contains `item2` one or more times.

Works with strings, lists, and anything that supports Python’s ‘in’ keyword. See `Should Be Equal` for an explanation on how to override the default error message with `msg` and `values`.

Examples: | Should Not Contain | ${output} | FAILED | | Should Not Contain | ${some_list} | value |

classmethod **contains** *(item1, item2, msg=None, values=True)*
Fails if `item1` does not contain `item2` one or more times.

Works with strings, lists, and anything that supports Python’s ‘in’ keyword. See `Should Be Equal` for an explanation on how to override the default error message with `msg` and `values`.

Examples: | Should Contain | ${output} | PASS | | Should Contain | ${some_list} | value |
classmethod `fail_unless_contains` *(item1, item2, msg=None, values=True)*
Fails if `item1` does not contain `item2` one or more times.

Works with strings, lists, and anything that supports Python’s ‘in’ keyword. See *Should Be Equal* for an explanation on how to override the default error message with `msg` and `values`.

Examples: | Should Contain | ${output} | PASS | | Should Contain | ${some_list} | value |

classmethod `does_not_match` *(string, pattern, msg=None, values=True)*
Fails if the given `string` matches the given `pattern`.

Pattern matching is similar as matching files in a shell, and it is always case-sensitive. In the pattern ‘*’ matches to anything and ‘?’ matches to any single character.

See *Should Be Equal* for an explanation on how to override the default error message with `msg` and `values`.

classmethod `fail_if_matches` *(string, pattern, msg=None, values=True)*
Fails if the given `string` matches the given `pattern`.

Pattern matching is similar as matching files in a shell, and it is always case-sensitive. In the pattern ‘*’ matches to anything and ‘?’ matches to any single character.

See *Should Be Equal* for an explanation on how to override the default error message with `msg` and `values`.

classmethod `matches` *(string, pattern, msg=None, values=True)*
Fails unless the given `string` matches the given `pattern`.

Pattern matching is similar as matching files in a shell, and it is always case-sensitive. In the pattern, ‘*’ matches to anything and ‘?’ matches to any single character.

See *Should Be Equal* for an explanation on how to override the default error message with `msg` and `values`.

classmethod `fail_unless_matches` *(string, pattern, msg=None, values=True)*
Fails unless the given `string` matches the given `pattern`.

Pattern matching is similar as matching files in a shell, and it is always case-sensitive. In the pattern, ‘*’ matches to anything and ‘?’ matches to any single character.

See *Should Be Equal* for an explanation on how to override the default error message with `msg` and `values`.

classmethod `does_not_match_regexp` *(string, pattern, msg=None, values=True)*
Fails if `string` matches `pattern` as a regular expression.

See *Should Match Regexp* for more information about arguments.

classmethod `fail_if_regexp_matches` *(string, pattern, msg=None, values=True)*
Fails if `string` matches `pattern` as a regular expression.

See *Should Match Regexp* for more information about arguments.

classmethod `matches_regexp` *(string, pattern, msg=None, values=True)*
Fails if `string` does not match `pattern` as a regular expression.

Regular expression check is done using the Python ‘re’ module, which has a pattern syntax derived from Perl, and thus also very similar to the one in Java. See the following documents for more details about regular expressions in general and Python implementation in particular.

http://docs.python.org/lib/module-re.html
http://www.amk.ca/python/howto/regex/

Things to note about the regexp syntax in Robot Framework test data:
1) Backslash is an escape character in the test data, and possible backslashes in the pattern must thus be escaped with another backslash (e.g. ‘\d\w+’).

2) Strings that may contain special characters, but should be handled as literal strings, can be escaped with the `Regexp Escape` keyword.

3) The given pattern does not need to match the whole string. For example, the pattern ‘ello’ matches the string ‘Hello world!’’. If a full match is needed, the ‘^’ and ‘$’ characters can be used to denote the beginning and end of the string, respectively. For example, ‘^ello$’ only matches the exact string ‘ello’.

4) Possible flags altering how the expression is parsed (e.g. `re.IGNORECASE`, `re.MULTILINE`) can be set by prefixing the pattern with the ‘(?iLmsux)’ group (e.g. ‘(?im)pattern’). The available flags are ‘IGNORECASE’: ‘i’, ‘MULTILINE’: ‘m’, ‘DOTALL’: ‘s’, ‘VERBOSE’: ‘x’, ‘UNICODE’: ‘u’, and ‘LOCALE’: ‘L’.

If this keyword passes, it returns the portion of the string that matched the pattern. Additionally, the possible captured groups are returned. See the `Should Be Equal` keyword for an explanation on how to override the default error message with the `msg` and `values` arguments.

Examples:
| Should Match Regexp | ${output} | \d{6} | # Output contains six numbers |
| Should Match Regexp | $({output}) | ^\d{6}$ | # Six numbers and nothing more |
| Should Match Regexp | Foo: 42 | (?i)foo: \d+ | |
| Should Match Regexp | Bar: 43 | (Foo|Bar): (\d+) | => |

```
class method fail_unless_regexp_matches (string, pattern, msg=None, values=True)
```

Fails if string does not match pattern as a regular expression.

Regular expression check is done using the Python ‘re’ module, which has a pattern syntax derived from Perl, and thus also very similar to the one in Java. See the following documents for more details about regular expressions in general and Python implementation in particular.

http://docs.python.org/lib/module-re.html
http://www.amk.ca/python/howto/regex/

Things to note about the regexp syntax in Robot Framework test data:

1) Backslash is an escape character in the test data, and possible backslashes in the pattern must thus be escaped with another backslash (e.g. ‘\d\w+’).

2) Strings that may contain special characters, but should be handled as literal strings, can be escaped with the `Regexp Escape` keyword.

3) The given pattern does not need to match the whole string. For example, the pattern ‘ello’ matches the string ‘Hello world!’’. If a full match is needed, the ‘^’ and ‘$’ characters can be used to denote the beginning and end of the string, respectively. For example, ‘^ello$’ only matches the exact string ‘ello’.

4) Possible flags altering how the expression is parsed (e.g. `re.IGNORECASE`, `re.MULTILINE`) can be set by prefixing the pattern with the ‘(?iLmsux)’ group (e.g. ‘(?im)pattern’). The available flags are ‘IGNORECASE’: ‘i’, ‘MULTILINE’: ‘m’, ‘DOTALL’: ‘s’, ‘VERBOSE’: ‘x’, ‘UNICODE’: ‘u’, and ‘LOCALE’: ‘L’.

If this keyword passes, it returns the portion of the string that matched the pattern. Additionally, the possible captured groups are returned. See the `Should Be Equal` keyword for an explanation on how to override the default error message with the `msg` and `values` arguments.
Examples: `Should Match Regexp | $\{output\} \ | \d{6} \ | # Output contains six numbers` `Should Match Regexp | $\{output\} \ | \d{6}\$ | # Six numbers and nothing more` `Should Match Regexp | Foo: \d+ | $\{match\} | $\{group1\} | $\{group2\} = 11` `Should Match Regexp | Bar: 43 | (Foo|Bar): \d{1, 2} | \$\{ret\} = 'Foo: 42' | $\{match\} = 'Bar: 43' | $\{group1\} = 'Bar' | $\{group2\} = '43'`

```python
classmethod noop ()

Does absolutely nothing.
```

classmethod set_(_(*values)

Returns the given values which can then be assigned to a variables.

This keyword is mainly used for setting scalar variables. Additionally it can be used for converting a scalar variable containing a list to a list variable or to multiple scalar variables. It is recommended to use `Create List` when creating new lists.

Examples: `Set Variable | Hello, world! | $\{hi2\} = $\{hi\} | $\{var1\} | $\{var2\} = $\{var\} | $\{hi\} | $\{var\} | $\{list\} = $\{list\} | $\{list\} | $\{item\} | $\{item\} = $\{list\} | $\{list\} | $\{item\} | $\{item\} = $\{list\} | $\{list\} | $\{item\} | $\{item\} = $\{list\} | $\{list\} | $\{item\} | $\{item\} = $\{list\} | $\{list\}

Variables created with this keyword are available only in the scope where they are created. See Set Global Variable, Set Test Variable and Set Suite Variable for information on how to set variables so that they are available also in a larger scope.

```python
classmethod message (*messages)

Displays the given messages in the log file as keyword arguments.

This keyword does nothing with the arguments it receives, but as they are visible in the log, this keyword can be used to display simple messages. Given arguments are ignored so thoroughly that they can even contain non-existing variables. If you are interested about variable values, you can use the Log or Log Many keywords.
```

classmethod variable_exists (name, msg=None)

Fails unless the given variable exists within the current scope.

The name of the variable can be given either as a normal variable name (e.g. `$\{NAME\}`) or in escaped format (e.g. `$\{NAME\}`). Notice that the former has some limitations explained in Set Suite Variable.

The default error message can be overridden with the `msg` argument.

See also Variable Should Not Exist and Keyword Should Exist.

```python
classmethod fail_unless_variable_exists (name, msg=None)

Fails unless the given variable exists within the current scope.

The name of the variable can be given either as a normal variable name (e.g. `$\{NAME\}`) or in escaped format (e.g. `$\{NAME\}`). Notice that the former has some limitations explained in Set Suite Variable.

The default error message can be overridden with the `msg` argument.

See also Variable Should Not Exist and Keyword Should Exist.
```

classmethod variable_does_not_exist (name, msg=None)

Fails if the given variable exists within the current scope.

The name of the variable can be given either as a normal variable name (e.g. `$\{NAME\}`) or in escaped format (e.g. `$\{NAME\}`). Notice that the former has some limitations explained in Set Suite Variable.

The default error message can be overridden with the `msg` argument.

See also Variable Should Exist and Keyword Should Exist.

```python
classmethod fail_if_variable_exists (name, msg=None)

Fails if the given variable exists within the current scope.
```

Examples: `Should Match Regexp | $\{output\} \ | \d{6} \ | # Output contains six numbers` `Should Match Regexp | $\{output\} \ | \d{6}\$ | # Six numbers and nothing more` `Should Match Regexp | Foo: \d+ | $\{match\} | $\{group1\} | $\{group2\} = 11` `Should Match Regexp | Bar: 43 | (Foo|Bar): \d{1, 2} | \$\{ret\} = 'Foo: 42' | $\{match\} = 'Bar: 43' | $\{group1\} = 'Bar' | $\{group2\} = '43'`
The name of the variable can be given either as a normal variable name (e.g. \$\{NAME\}) or in escaped format (e.g. \$\{NAME\}). Notice that the former has some limitations explained in Set Suite Variable.

The default error message can be overridden with the \texttt{msg} argument.
See also Variable Should Exist and Keyword Should Exist.

\texttt{error}(msg=None)

Errors the test immediately with the given message.

\texttt{grep}(text, pattern, pattern\_type='literal string')

\begin{verbatim}
DeprecatedOperatingSystem Module
\end{verbatim}

\begin{verbatim}
class robot.libraries.DeprecatedOperatingSystem.DeprecatedOperatingSystem

ROBOT_LIBRARY_SCOPE = 'GLOBAL'

classmethod delete_environment_variable(*names)

Deletes the specified environment variable.

Does nothing if the environment variable is not set.

Starting from Robot Framework 2.7, it is possible to remove multiple variables by passing them to this keyword as separate arguments.

classmethod environment_variable_is_set(name, msg=None)

Fails if the specified environment variable is not set.

The default error message can be overridden with the \texttt{msg} argument.

classmethod environment_variable_is_not_set(name, msg=None)

Fails if the specified environment variable is set.

The default error message can be overridden with the \texttt{msg} argument.

classmethod fail_unless_exists(path, msg=None)

Fails unless the given path (file or directory) exists.

The path can be given as an exact path or as a glob pattern. The pattern matching syntax is explained in introduction. The default error message can be overridden with the \texttt{msg} argument.

classmethod fail_if_exists(path, msg=None)

Fails if the given path (file or directory) exists.

The path can be given as an exact path or as a glob pattern. The pattern matching syntax is explained in introduction. The default error message can be overridden with the \texttt{msg} argument.

classmethod fail_unless_file_exists(path, msg=None)

Fails unless the given path points to an existing file.

The path can be given as an exact path or as a glob pattern. The pattern matching syntax is explained in introduction. The default error message can be overridden with the \texttt{msg} argument.

classmethod fail_if_file_exists(path, msg=None)

Fails if the given path points to an existing file.

The path can be given as an exact path or as a glob pattern. The pattern matching syntax is explained in introduction. The default error message can be overridden with the \texttt{msg} argument.

classmethod fail_unless_dir_exists(path, msg=None)

Fails unless the given path points to an existing directory.
\end{verbatim}
The path can be given as an exact path or as a glob pattern. The pattern matching syntax is explained in *introduction*. The default error message can be overridden with the `msg` argument.

```python
classmethod fail_if_dir_exists (path, msg=None)
    Fails if the given path points to an existing file.
```

The path can be given as an exact path or as a glob pattern. The pattern matching syntax is explained in *introduction*. The default error message can be overridden with the `msg` argument.

```python
classmethod fail_unless_dir_empty (path, msg=None)
    Fails unless the specified directory is empty.
```

The default error message can be overridden with the `msg` argument.

```python
classmethod fail_if_dir_empty (path, msg=None)
    Fails if the specified directory is empty.
```

The default error message can be overridden with the `msg` argument.

```python
classmethod fail_unless_file_empty (path, msg=None)
    Fails unless the specified file is empty.
```

The default error message can be overridden with the `msg` argument.

```python
classmethod fail_if_file_empty (path, msg=None)
    Fails if the specified directory is empty.
```

The default error message can be overridden with the `msg` argument.

```python
classmethod empty_dir (path)
    Deletes all the content (incl. subdirectories) from the given directory.
```

```python
classmethod remove_dir (path, recursive=False)
    Removes the directory pointed to by the given `path`.
    If the second argument `recursive` is set to any non-empty string, the directory is removed recursively.
    Otherwise removing fails if the directory is not empty.
    If the directory pointed to by the `path` does not exist, the keyword passes, but it fails, if the `path` points to a file.
```

```python
classmethod copy_dir (source, destination)
    Copies the source directory into the destination.
    If the destination exists, the source is copied under it. Otherwise the destination directory and the possible missing intermediate directories are created.
```

```python
classmethod move_dir (source, destination)
    Moves the source directory into a destination.
    Uses Copy Directory keyword internally, and `source` and `destination` arguments have exactly same semantics as with that keyword.
```

```python
classmethod create_dir (path)
    Creates the specified directory.
    Also possible intermediate directories are created. Passes if the directory already exists, and fails if the path points to a regular file.
```

```python
classmethod list_dir (path, pattern=None, absolute=False)
    Returns and logs items in a directory, optionally filtered with `pattern`.
    File and directory names are returned in case-sensitive alphabetical order, e.g. ['A Name', 'Second', 'a lower case name', 'one more']. Implicit directories '.' and '..' are not returned. The returned items are automatically logged.
```
By default, the file and directory names are returned relative to the given path (e.g. ‘file.txt’). If you want them be returned in the absolute format (e.g. ‘/home/robot/file.txt’), set the absolute argument to any non-empty string.

If pattern is given, only items matching it are returned. The pattern matching syntax is explained in introduction, and in this case matching is case-sensitive.

Examples (using also other List Directory variants):  
| @{items} = | List Directory | ${TEMPDIR} | |  
| @{files} = | List Files In Directory | /tmp | *.txt | absolute | | ${count} = | Count Files In Directory | ${CURDIR} | ??? |

classmethod list_files_in_dir(path, pattern=None, absolute=False)  
A wrapper for List Directory that returns only files.

classmethod list_dirs_in_dir(path, pattern=None, absolute=False)  
A wrapper for List Directory that returns only directories.

classmethod count_items_in_dir(path, pattern=None)  
Returns and logs the number of all items in the given directory.

The argument pattern has the same semantics as in the List Directory keyword. The count is returned as an integer, so it must be checked e.g. with the built-in keyword Should Be Equal As Integers.

classmethod count_files_in_dir(path, pattern=None)  
A wrapper for Count Items In Directory returning only file count.

classmethod count_dirs_in_dir(path, pattern=None)  
A wrapper for Count Items In Directory returning only directory count.

**Dialogs Module**

A test library providing dialogs for interacting with users.

**Dialogs** is Robot Framework’s standard library that provides means for pausing the test execution and getting input from users. The dialogs are slightly different depending on are tests run on Python or Jython but they provide the same functionality.

The library has following two limitations: - It is not compatible with IronPython. - It cannot be used with timeouts on Python.

```python
robot.libraries.Dialogs.pause_execution(message='Test execution paused. Press OK to continue."
```
Pauses test execution until user clicks Ok button.

message is the message shown in the dialog.

```python
robot.libraries.Dialogs.execute_manual_step(message, default_error='')
```
Pauses test execution until user sets the keyword status.

User can select ‘PASS’ or ‘FAIL’, and in the latter case an additional dialog is opened for defining the error message.

message is the instruction shown in the initial dialog and default_error is the default value shown in the possible error message dialog.

```python
robot.libraries.Dialogs.get_value_from_user(message, default_value='')
```
Pauses test execution and asks user to input a value.

message is the instruction shown in the dialog and default_value is the possible default value shown in the input field. Selecting ‘Cancel’ fails the keyword.
robot.libraries.Dialogs.get_selection_from_user(message, *values)
Pauses test execution and asks user to select a value.
message is the instruction shown in the dialog and values are the options given to the user. Selecting ‘Cancel’ fails the keyword.

Easter Module

robot.libraries.Easter.none_shall_pass(who)

OperatingSystem Module

class robot.libraries.OperatingSystem.OperatingSystem
A test library providing keywords for OS related tasks.

OperatingSystem is Robot Framework’s standard library that enables various operating system related tasks to be performed in the system where Robot Framework is running. It can, among other things, execute commands (e.g. Run), create and remove files and directories (e.g. Create File, Remove Directory), check whether files or directories exists or contain something (e.g. File Should Exist, Directory Should Be Empty) and manipulate environment variables (e.g. Set Environment Variable).

= Pattern matching =
Some keywords allow their arguments to be specified as _glob_ patterns where: | * | matches anything, even an empty string || ? | matches any single character || [chars] | matches any character inside square brackets (e.g. ’[abc]’ matches either ’a’, ’b’ or ’c’) || [!chars] | matches any character not inside square brackets |
Unless otherwise noted, matching is case-insensitive on case-insensitive operating systems such as Windows. Pattern matching is implemented using Python’s fnmatch module: http://docs.python.org/library/fnmatch.html

= Path separators =
All keywords expecting paths as arguments accept a forward slash (/) as a path separator regardless the operating system. Notice that this does not work when the path is part of an argument, like it often is with Run and Start Process keywords. In such cases the built-in variable ${/} can be used to keep the test data platform independent.

= Example =

| Setting | Value |
Library | OperatingSystem |

| Variable | Value |
${PATH} | ${CURDIR}/example.txt |

| Test Case | Action | Argument | Argument |
Example | Create File | ${PATH} | Some text |
| File Should Exist | ${PATH} |
| Copy File | ${PATH} | ${TEMPDIR}/stuff |
| ${output} = | Run | ${CURDIR}$/script.py arg |

ROBOT_LIBRARY_SCOPE = ‘GLOBAL’
run (command)
Runs the given command in the system and returns the output.

The execution status of the command is not checked by this keyword, and it must be done separately based on the returned output. If the execution return code is needed, either Run And Return RC or Run And Return RC And Output can be used.

The standard error stream is automatically redirected to the standard output stream by adding 2>&1 after the executed command. This automatic redirection is done only when the executed command does not contain additional output redirections. You can thus freely forward the standard error somewhere else, for example, like my_command 2>stderr.txt.

The returned output contains everything written into the standard output or error streams by the command (unless either of them is redirected explicitly). Many commands add an extra newline (\n) after the output to make it easier to read in the console. To ease processing the returned output, this possible trailing newline is stripped by this keyword.

Examples:

```
| ${output} = | Run | ls -lhF /tmp | | Log | ${output} | | ${result} = | Run | ${CURDIR}/${}/tester.py arg1 arg2 | Should Not Contain | ${result} | FAIL | | $stdout = | Run | /opt/script.sh 2>/tmp/stderr.txt | Should Be Equal | $stdout | TEST PASSED | File Should Be Empty | /tmp/stderr.txt |
```

run_and_return_rc (command)
Runs the given command in the system and returns the return code.

The return code (RC) is returned as a positive integer in range from 0 to 255 as returned by the executed command. On some operating systems (notable Windows) original return codes can be something else, but this keyword always maps them to the 0-255 range. Since the RC is an integer, it must be checked e.g. with the keyword Should Be Equal As Integers instead of Should Be Equal (both are built-in keywords).

Examples:

```
| ${rc} = | Run and Return RC | ${CURDIR}/${}/script.py arg1 Should Be Equal As Integers | \$rc | 0 | | ${rc} = | Run and Return RC | /path/to/example.rb arg1 arg2 | Should Be True | 0 < $rc < 42 |
```

See Run and Run And Return RC And Output if you need to get the output of the executed command.

run_and_return_rc_and_output (command)
Runs the given command in the system and returns the RC and output.

The return code (RC) is returned similarly as with Run And Return RC and the output similarly as with Run.

Examples:

```
| $rc | $output = | Run and Return RC and Output | ${CURDIR}/${}/mytool | Should Be Equal As Integers | $rc | 0 | Should Not Contain | $output | FAIL | $rc | $stdout = | Run and Return RC and Output | /opt/script.sh 2>/tmp/stderr.txt | Should Be True | $rc > 42 | Should Be Equal | $stdout | TEST PASSED | File Should Be Empty | /tmp/stderr.txt |
```

start_process (command, stdin=None, alias=None)
Starts the given command as a background process.

Starts the process in background and sets it as the active process. Read Process Output or Stop Process keywords affect this process unless Switch Process is used in between.

If the command needs input through the standard input stream, it can be defined with the stdin argument. It is not possible to give input to the command later. Possible command line arguments must be given as part of the command like ‘/tmp/script.sh arg1 arg2’.

Returns the index of this process. Indexing starts from 1, and indices can be used to switch between processes using Switch Process keyword. Stop All Processes can be used to reset indexing.
The optional alias is a name for this process that may be used with Switch Process instead of the returned index.

The standard error stream is redirected to the standard input stream automatically. This is done for the same reasons as with Run keyword, but redirecting is done when the process is started and not by adding ‘2>&1’ to the command.

Example: | Start Process | /path/longlasting.sh | | Do Something | | $\{\text{output}\} = | Read Process Output | |
Should Contain | $\{\text{output}\} | Expected text | | [Teardown] | Stop All Processes |

**switch_process(index_or_alias)**

Switches the active process to the specified process.

New active process can be specified either using an index or an alias. Indices are return values from Start Process and aliases can be given to that keyword.


**read_process_output()**

Waits for a process to finish and returns its output.

This keyword waits for a process started with Start Process to end and then returns all output it has produced. The returned output contains everything the process has written into the standard output and error streams.

There is no need to use Stop Process after using this keyword. Trying to read from an already stopped process fails.

Note that although the process is finished, it still stays as the active process. Use Switch Process to switch the active process or Stop All Processes to reset the list of started processes.

**stop_process()**

Closes the standard output stream of the process.

This keyword does not actually stop the process nor even wait for it to terminate. Only thing it does is closing the standard output stream of the process. Depending on the process that may terminate it but that is not guaranteed. Use Read Process Output instead if you need to wait for the process to complete.

This keyword operates the active process similarly as Read Process Output. Stopping an already stopped process is not an error.

**stop_all_processes()**

Closes the standard output of all the processes and resets the process list.

Exactly like Stop Process, this keyword does not actually stop processes nor even wait for them to terminate.

This keyword resets the indexing that Start Process uses. All aliases are also deleted. It does not matter have some of the processes already been closed or not.

**get_file(path, encoding=’UTF-8’)**

Returns the contents of a specified file.

This keyword reads the specified file and returns the contents. Line breaks in content are converted to platform independent form. See also Get Binary File.

encoding defines the encoding of the file. By default the value is ‘UTF-8’, which means that UTF-8 and ASCII-encoded files are read correctly.
get_binary_file(path)
Returns the contents of a specified file.

This keyword reads the specified file and returns the contents as is. See also Get File.

New in Robot Framework 2.5.5.

grep_file(path, pattern, encoding='UTF-8')
Returns the lines of the specified file that match the pattern.

This keyword reads a file from the file system using the defined path and encoding similarly as Get File. A difference is that only the lines that match the given pattern are returned. Lines are returned as a single string catenated back together with newlines and the number of matched lines is automatically logged. Possible trailing newline is never returned.

A line matches if it contains the pattern anywhere in it and it does not need to match the pattern fully. The pattern matching syntax is explained in introduction, and in this case matching is case-sensitive.

Examples: | ${errors} = | Grep File | /var/log/myapp.log | ERROR | | ${ret} = | Grep File | ${CUR-DIR}/file.txt | [Ww]ildc??d ex*ple |

If more complex pattern matching is needed, it is possible to use Get File in combination with String library keywords like Get Lines Matching Regexp.

log_file(path, encoding='UTF-8')
Wrapper for Get File that also logs the returned file.

The file is logged with the INFO level. If you want something else, just use Get File and the built-in keyword Log with the desired level.

should_exist(path, msg=None)
Fails unless the given path (file or directory) exists.

The path can be given as an exact path or as a glob pattern. The pattern matching syntax is explained in introduction. The default error message can be overridden with the msg argument.

should_not_exist(path, msg=None)
Fails if the given path (file or directory) exists.

The path can be given as an exact path or as a glob pattern. The pattern matching syntax is explained in introduction. The default error message can be overridden with the msg argument.

file_should_exist(path, msg=None)
Fails unless the given path points to an existing file.

The path can be given as an exact path or as a glob pattern. The pattern matching syntax is explained in introduction. The default error message can be overridden with the msg argument.

file_should_not_exist(path, msg=None)
Fails if the given path points to an existing file.

The path can be given as an exact path or as a glob pattern. The pattern matching syntax is explained in introduction. The default error message can be overridden with the msg argument.

directory_should_exist(path, msg=None)
Fails unless the given path points to an existing directory.

The path can be given as an exact path or as a glob pattern. The pattern matching syntax is explained in introduction. The default error message can be overridden with the msg argument.

directory_should_not_exist(path, msg=None)
Fails if the given path points to an existing file.
The path can be given as an exact path or as a glob pattern. The pattern matching syntax is explained in
`introduction`. The default error message can be overridden with the `msg` argument.

**wait_until_removed(path, timeout='1 minute')**

Waits until the given file or directory is removed.

The path can be given as an exact path or as a glob pattern. The pattern matching syntax is explained in
`introduction`. If the path is a pattern, the keyword waits until all matching items are removed.

The optional `timeout` can be used to control the maximum time of waiting. The timeout is given as a
timeout string, e.g. in a format ‘15 seconds’, ‘1min 10s’ or just ‘10’. The time string format is described

If the timeout is negative, the keyword is never timed-out. The keyword returns immediately, if the path
does not exist in the first place.

**wait_until_created(path, timeout='1 minute')**

Waits until the given file or directory is created.

The path can be given as an exact path or as a glob pattern. The pattern matching syntax is explained in
`introduction`. If the path is a pattern, the keyword returns when an item matching it is created.

The optional `timeout` can be used to control the maximum time of waiting. The timeout is given as a
timeout string, e.g. in a format ‘15 seconds’, ‘1min 10s’ or just ‘10’. The time string format is described

If the timeout is negative, the keyword is never timed-out. The keyword returns immediately, if the path
already exists.

**directory_should_be_empty(path, msg=None)**

Fails unless the specified directory is empty.

The default error message can be overridden with the `msg` argument.

**directory_should_not_be_empty(path, msg=None)**

Fails if the specified directory is empty.

The default error message can be overridden with the `msg` argument.

**file_should_be_empty(path, msg=None)**

Fails unless the specified file is empty.

The default error message can be overridden with the `msg` argument.

**file_should_not_be_empty(path, msg=None)**

Fails if the specified directory is empty.

The default error message can be overridden with the `msg` argument.

**create_file(path, content='', encoding='UTF-8')**

Creates a file with the given content and encoding.

If the directory where to create file does not exist it, and possible intermediate missing directories, are
created.

Use `Append To File` if you want to append to an existing file, and use `File Should Not Exist` if you want to
avoid overwriting existing files.

**append_to_file(path, content, encoding='UTF-8')**

Appends the given contend to the specified file.

If the file does not exists, this keyword works exactly the same way as `Create File With Encoding`. 
**remove_file**(path)
Removes a file with the given path.

Passes if the file does not exist, but fails if the path does not point to a regular file (e.g. it points to a directory).

The path can be given as an exact path or as a glob pattern. The pattern matching syntax is explained in *introduction*. If the path is a pattern, all files matching it are removed.

**remove_files**(paths)
Uses *Remove File* to remove multiple files one-by-one.

Example:  |
| Remove Files | ${TEMPDIR}${/}foo.txt | ${TEMPDIR}${/}bar.txt | ${TEMPDIR}${/}zap.txt |

**empty_directory**(path)
Deletes all the content (incl. subdirectories) from the given directory.

**create_directory**(path)
Creates the specified directory.

Also possible intermediate directories are created. Passes if the directory already exists, and fails if the path points to a regular file.

**remove_directory**(path, recursive=False)
Removes the directory pointed to by the given path.

If the second argument recursive is set to any non-empty string, the directory is removed recursively. Otherwise removing fails if the directory is not empty.

If the directory pointed to by the path does not exist, the keyword passes, but it fails, if the path points to a file.

**copy_file**(source, destination)
Copies the source file into a new destination.

1) If the destination is an existing file, the source file is copied over it.

2) If the destination is an existing directory, the source file is copied into it. A possible file with the same name is overwritten.

3) If the destination does not exist and it ends with a path separator (‘/’ or ‘\’), it is considered a directory. That directory is created and a source file copied into it. Possible missing intermediate directories are also created.

4) If the destination does not exist and it does not end with a path separator, it is considered a file. If the path to the file does not exist, it is created.

**move_file**(source, destination)
Moves the source file into a new destination.

Uses *Copy File* keyword internally, and source and destination arguments have exactly same semantics as with that keyword.

**copy_directory**(source, destination)
Copies the source directory into the destination.

If the destination exists, the source is copied under it. Otherwise the destination directory and the possible missing intermediate directories are created.

**move_directory**(source, destination)
Moves the source directory into a destination.
Uses *Copy Directory* keyword internally, and *source* and *destination* arguments have exactly same semantics as with that keyword.

**get_environment_variable** (name, default=None)

Returns the value of an environment variable with the given name.

If no such environment variable is set, returns the default value, if given. Otherwise fails the test case.

Starting from Robot Framework 2.7, returned variables are automatically decoded to Unicode using the system encoding.

Note that you can also access environment variables directly using the variable syntax `%{ENV_VAR_NAME}`.

**set_environment_variable** (name, value)

Sets an environment variable to a specified value.

Values are converted to strings automatically. Starting from Robot Framework 2.7, set variables are automatically encoded using the system encoding.

**remove_environment_variable** (*names*)

Deletes the specified environment variable.

Does nothing if the environment variable is not set.

Starting from Robot Framework 2.7, it is possible to remove multiple variables by passing them to this keyword as separate arguments.

**environment_variable_should_be_set** (name, msg=None)

Fails if the specified environment variable is not set.

The default error message can be overridden with the *msg* argument.

**environment_variable_should_not_be_set** (name, msg=None)

Fails if the specified environment variable is set.

The default error message can be overridden with the *msg* argument.

**get_environment_variables** ()

Returns currently available environment variables as a dictionary.

Both keys and values are decoded to Unicode using the system encoding. Altering the returned dictionary has no effect on the actual environment variables.

New in Robot Framework 2.7.

**log_environment_variables** (level='INFO')

Logs all environment variables using the given log level.

Environment variables are also returned the same way as with *Get Environment Variables* keyword.

New in Robot Framework 2.7.

**join_path** (base, *parts)

Joins the given path part(s) to the given base path.

The path separator ('/' or '\') is inserted when needed and the possible absolute paths handled as expected. The resulted path is also normalized.

Examples:

| $path | J o i n P a t h | m y | path | $p2 | J o i n P a t h | m y/ | path/ | $p3 | J o i n P a t h | m y/ | path/ | fi l e . t x t | $p4 | J o i n P a t h | m y/ | path/ | .. | path2 | |
|-------|----------------|------|------|-----|----------------|------|------|-----|----------------|------|------|----------------|------|------|----------------|------|------|----------------|------|------|----------------|
| =>    | - $(path) = 'my/path' - $p2 = 'my/path' - $p3 = 'my/path/my/file.txt' - $p4 = '/path' - $p5 = '/my/path2' |
### join_paths (base, *paths)

Joins given paths with base and returns resulted paths.

See Join Path for more information.

Examples:

```robot
@ {p1} = Join Path base example other \| @ {p2} = Join Path /my/base example/other \| @ {p3} = Join Path /my/base/example/other=\=> - @ {p1} = ['base/example', 'base/other'] - @ {p2} = ['/example', '/my/base/other'] - @ {p3} = ['my/base/example/path', 'my/base/other', 'my/base/one/more']
```

### normalize_path (path)

Normalizes the given path.

Examples:

```robot
$ {path} = Normalize Path abc \| $ {p2} = Normalize Path abc/ \| $ {p3} = Normalize Path abc//def \| => - $ {path} = 'abc' - $ {p2} = 'abc' - $ {p3} = 'def' - $ {p4} = 'abc/def' - $ {p5} = 'abc/
```

### split_path (path)

Splits the given path from the last path separator ('/' or '\').

The given path is first normalized (e.g. a possible trailing path separator is removed, special directories '..' and '.' removed). The parts that are split are returned as separate components.

Examples:

```robot
$ {path1} \| $ {dir} = Split Path abc/def \| $ {file} = Split Path abc/def/ghi.txt \| $ {d2} = Split Path abc/../def/ghi=\=> - $ {path1} = 'abc' & $ {dir} = 'def' - $ {path2} = 'def' & $ {file} = 'ghi.txt' - $ {path3} = 'def' & $ {d2} = 'ghi'
```

### split_extension (path)

Splits the extension from the given path.

The given path is first normalized (e.g. possible trailing path separators removed, special directories '..' and '.' removed). The base path and extension are returned as separate components so that the dot used as an extension separator is removed. If the path contains no extension, an empty string is returned for it. Possible leading and trailing dots in the file name are never considered to be extension separators.

Examples:

```robot
$ {path} \| $ {ext} = Split Extension file.extension \| $ {e2} = Split Extension p1/..p2/file.ext \| => - $ {path} = 'file' & $ {ext} = 'extension' - $ {p2} = 'p2/file' & $ {e2} = 'ext' - $ {e3} = '' - $ {p4} = 'p2/file' & $ {e4} = 'ext' - $ {p5} = 'path/file' & $ {e5} = 'ext' - $ {p6} = 'path/file' & $ {e6} = ''
```

### get_modified_time (path, format='timestamp')

Returns the last modification time of a file or directory.

How time is returned is determined based on the given format string as follows. Note that all checks are case-insensitive. Returned time is also automatically logged.

1. If format contains the word 'epoch', the time is returned in seconds after the UNIX epoch. The return value is always an integer.

2. If format contains any of the words 'year', 'month', 'day', 'hour', 'min' or 'sec', only the selected parts are returned. The order of the returned parts is always the one in the previous sentence and the order of the words in format is not significant. The parts are returned as zero-padded strings (e.g. May -> '05').

3. Otherwise, and by default, the time is returned as a timestamp string in the format '2006-02-24 15:08:31'.

Examples (when the modified time of the $CURDIR is 2006-03-29 15:06:21):

```robot
$ {time} = Get Modified Time \| $ {time} = Get Modified Time $CURDIR \| $ {time} = Get Modified Time $CURDIR \| epoch \| $ {time} = epoch \| $ {time} = Get Modified Time $CURDIR \| return year \| $ {time} = return year,day
```
set_modified_time (path, mtime)

Sets the file modification and access times.

Changes the modification and access times of the given file to the value determined by `mtime`. The time can be given in different formats described below. Note that all checks involving strings are case-insensitive.

1. If `mtime` is a number, or a string that can be converted to a number, it is interpreted as seconds since the UNIX epoch (1970-01-01 00:00:00 UTC). This documentation was originally written about 1177654467 seconds after the epoch.

2. If `mtime` is a timestamp, that time will be used. Valid timestamp formats are ‘YYYY-MM-DD hh:mm:ss’ and ‘YYYYMMDD hhmmss’.

3. If `mtime` is equal to ‘NOW’, the current local time is used. This time is got using Python’s `time.time()` function.

4. If `mtime` is equal to ‘UTC’, the current time in [http://en.wikipedia.org/wiki/Coordinated_Universal_Time|UTC] is used. This time is got using `time.time() + time.altzone` in Python.

5. If `mtime` is in the format like ‘NOW - 1 day’ or ‘UTC + 1 hour 30 min’, the current local/UTC time plus/minus the time specified with the time string is used. The time string format is described in an appendix of Robot Framework User Guide.

Examples:

| Set Modified Time | /path/file | 1177654467 | # Time given as epoch seconds |
| Set Modified Time | /path/file | 2007-04-27 9:14:27 | # Time given as a timestamp |
| Set Modified Time | /path/file | NOW | # The local time of execution |
| Set Modified Time | /path/file | UTC + 1h 2min 3s | # UTC time plus 1h 2min 3s |
| Set Modified Time | /path/file | UTC - 1 day | # 1 day subtracted from local time |

Support for UTC time is a new feature in Robot Framework 2.7.5.

get_file_size (path)

Returns and logs file size as an integer in bytes

list_directory (path, pattern=None, absolute=False)

Returns and logs items in a directory, optionally filtered with `pattern`.

File and directory names are returned in case-sensitive alphabetical order, e.g. ‘A Name’, ‘Second’, ‘a lower case name’, ‘one more’]. Implicit directories ‘.’ and ‘..’ are not returned. The returned items are automatically logged.

By default, the file and directory names are returned relative to the given path (e.g. ‘file.txt’). If you want them be returned in the absolute format (e.g. ‘/home/robot/file.txt’), set the `absolute` argument to any non-empty string.

If `pattern` is given, only items matching it are returned. The pattern matching syntax is explained in `Introduction`, and in this case matching is case-sensitive.

Examples (using also other `List Directory` variants): |

| List Files In Directory | /tmp | *txt | absolute | # Count Files In Directory |

list_files_in_directory (path, pattern=None, absolute=False)

A wrapper for `List Directory` that returns only files.

list_directories_in_directory (path, pattern=None, absolute=False)

A wrapper for `List Directory` that returns only directories.
count_items_in_directory (path, pattern=None)
    Returns and logs the number of all items in the given directory.

    The argument pattern has the same semantics as in the List Directory keyword. The count is returned as
    an integer, so it must be checked e.g. with the built-in keyword Should Be Equal As Integers.

count_files_in_directory (path, pattern=None)
    A wrapper for Count Items In Directory returning only file count.

count_directories_in_directory (path, pattern=None)
    A wrapper for Count Items In Directory returning only directory count.
touch (path)
    Emulates the UNIX touch command.

    Creates a file, if it does not exist. Otherwise changes its access and modification times to the current time.

    Fails if used with the directories or the parent directory of the given file does not exist.

Remote Module

    ROBOT_LIBRARY_SCOPE = 'TEST SUITE'
    get_keyword_names (attempts=5)
    get_keyword_arguments (name)
    get_keyword_documentation (name)
    run_keyword (name, args)

class robot.libraries.Remote.RemoteResult (result)

class robot.libraries.Remote.XmlRpcRemoteClient (uri)

    get_keyword_names ()
    get_keyword_arguments (name)
    get_keyword_documentation (name)
    run_keyword (name, args)

Reserved Module
class robot.libraries.Reserved.Reserved

    ROBOT_LIBRARY_SCOPE = 'GLOBAL'
    get_keyword_names ()
    run_keyword (name, args)
Screenshot Module

```python
class robot.libraries.Screenshot.Screenshot (screenshot_directory=None)
    Bases: object

Test library for taking screenshots on the machine where tests are run.

Notice that successfully taking screenshots requires tests to be run with a physical or virtual display.

Using with Python

With Python you need to have one of the following modules installed to be able to use this library. The first module that is found will be used.

• wxPython :: http://wxpython.org :: Required also by RIDE so many Robot Framework users already have this module installed.
• PyGTK :: http://pygtk.org :: This module is available by default on most Linux distributions.
• Python Imaging Library (PIL) :: http://www.pythonware.com/products/pil :: This module can take screenshots only on Windows.

Python support was added in Robot Framework 2.5.5.

Using with Jython and IronPython

With Jython and IronPython this library uses APIs provided by JVM and .NET platforms, respectively. These APIs are always available and thus no external modules are needed.

IronPython support was added in Robot Framework 2.7.5.

Where screenshots are saved

By default screenshots are saved into the same directory where the Robot Framework log file is written. If no log is created, screenshots are saved into the directory where the XML output file is written.

It is possible to specify a custom location for screenshots using

`screenshot_directory` argument in importing and `Set Screenshot Directory` keyword during execution. It is also possible to save screenshots using an absolute path.

Note that prior to Robot Framework 2.5.5 the default screenshot location was system’s temporary directory.

Changes in Robot Framework 2.5.5 and Robot Framework 2.6

This library was heavily enhanced in Robot Framework 2.5.5 release. The changes are listed below and explained more thoroughly in affected places.

• The support for using this library on Python (see above) was added.
• The default location where screenshots are saved was changed (see above).
• New `Take Screenshot` and `Take Screenshot Without Embedding` keywords were added. These keywords should be used for taking screenshots in the future. Other screenshot taking keywords will be deprecated and removed later.
• `log_file_directory` argument was deprecated everywhere it was used.

In Robot Framework 2.6, following additional changes were made:

• `log_file_directory` argument was removed altogether.
• `Set Screenshot Directories` keyword was removed.
• Save Screenshot, Save Screenshot To and Log Screenshot keywords were deprecated. They will be removed in Robot Framework 2.8.

Configure where screenshots are saved.

If screenshot_directory is not given, screenshots are saved into same directory as the log file. The directory can also be set using Set Screenshot Directory keyword.

Examples (use only one of these):

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
<th>Value</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library</td>
<td>Screenshot</td>
<td></td>
<td># Default location</td>
</tr>
<tr>
<td>Library</td>
<td>Screenshot</td>
<td>${TEMPDIR}</td>
<td># System temp (this was default prior to 2.5.5)</td>
</tr>
</tbody>
</table>

ROBOT_LIBRARY_SCOPE = ‘TEST SUITE’

ROBOT_LIBRARY_VERSION = ‘2.7.7’

set_screenshot_directory (path)
Sets the directory where screenshots are saved.

It is possible to use / as a path separator in all operating systems. Path to the old directory is returned.

The directory can also be set in importing.

save_screenshot_to (path)
DEPRECATED Use Take Screenshot or Take Screenshot Without Embedding instead.

save_screenshot (basename='screenshot', directory=None)
DEPRECATED Use Take Screenshot or Take Screenshot Without Embedding instead.

log_screenshot (basename='screenshot', directory=None, width='100%')
DEPRECATED Use Take Screenshot or Take Screenshot Without Embedding instead.

take_screenshot (name='screenshot', width='800px')
Takes a screenshot in JPEG format and embeds it into the log file.

Name of the file where the screenshot is stored is derived from the given name. If the name ends with extension .jpg or .jpeg, the screenshot will be stored with that exact name. Otherwise a unique name is created by adding an underscore, a running index and an extension to the name.

The name will be interpreted to be relative to the directory where the log file is written. It is also possible to use absolute paths. Using / as a path separator works in all operating systems.

width specifies the size of the screenshot in the log file.

Examples: (LOGDIR is determined automatically by the library) \| Take Screenshot \| \| \| # LOGDIR/screenrshot_1.jpg (index automatically incremented) \| \| Take Screenshot \| mypic \| \| # LOGDIR/mypic_1.jpg (index automatically incremented) \| \| Take Screenshot \| ${TEMPDIR}/mypic \| \# /tmp/mypic_1.jpg (index automatically incremented) \| \| Take Screenshot \| images/login.jpg 80% \| # Specify both name and width. \| \| Take Screenshot \| width=550px \| \# Specify only width. \|

The path where the screenshot is saved is returned.

take_screenshot_without_embedding (name='screenshot')
Takes a screenshot and links it from the log file.

This keyword is otherwise identical to Take Screenshot but the saved screenshot is not embedded into the log file. The screenshot is linked so it is nevertheless easily available.
class robot.libraries.Screenshot.ScreenshotTaker(module_name=None)
  Bases: object

String Module

class robot.libraries.String.String
  A test library for string manipulation and verification.

  String is Robot Framework's standard library for manipulating strings (e.g. Replace String Using Regexp, Split To Lines) and verifying their contents (e.g. Should Be String).

  Following keywords from BuiltIn library can also be used with strings:

  • Catenate
  • Get Length
  • Length Should Be
  • Should (Not) Be Empty
  • Should (Not) Be Equal (As Strings/Integers/Numbers)
  • Should (Not) Match (Regexp)
  • Should (Not) Contain
  • Should (Not) Start With
  • Should (Not) End With
  • Convert To String

  ROBOT_LIBRARY_SCOPE = 'GLOBAL'
  ROBOT_LIBRARY_VERSION = '2.7.7'

  encode_string_to_bytes(string, encoding, errors='strict')
  Encodes the given Unicode string to bytes using the given encoding.

  errors argument controls what to do if encoding some characters fails. All values accepted by encode method in Python are valid, but in practice the following values are most useful:

  • strict: fail if characters cannot be encoded (default)
  • ignore: ignore characters that cannot be encoded
  • replace: replace characters that cannot be encoded with a replacement character

  Examples: | ${bytes} = | Encode String To Bytes | ${string} | UTF-8 | | ${bytes} = | Encode String To Bytes | ${string} | ASCII | errors=ignore |

  Use Decode Bytes To String if you need to convert byte strings to Unicode strings, and Convert To String in BuiltIn if you need to convert arbitrary objects to Unicode strings.

  New in Robot Framework 2.7.7.

  decode_bytes_to_string(bytes, encoding, errors='strict')
  Decodes the given bytes to a Unicode string using the given encoding.

  errors argument controls what to do if decoding some bytes fails. All values accepted by decode method in Python are valid, but in practice the following values are most useful:

  • strict: fail if characters cannot be decoded (default)
  • ignore: ignore characters that cannot be decoded
•*replace*: replace characters that cannot be decoded with a replacement character

Examples: | ${string} = | Decode Bytes To String | ${bytes} | UTF-8 | | ${string} = | Decode Bytes To String | ${bytes} | ASCII | errors=ignore |

Use *Encode String To Bytes* if you need to convert Unicode strings to byte strings, and *Convert To String* in *BuiltIn* if you need to convert arbitrary objects to Unicode strings.

New in Robot Framework 2.7.7.

**get_line_count** *(string)*

Returns and logs the number of lines in the given *string*.

**split_to_lines** *(string, start=0, end=None)*

Converts the *string* into a list of lines.

It is possible to get only a selection of lines from *start* to *end* so that *start* index is inclusive and *end* is exclusive. Line numbering starts from 0, and it is possible to use negative indices to refer to lines from the end.

Lines are returned without the newlines. The number of returned lines is automatically logged.

Examples: | @|@{lines} = | Split To Lines | ${manylines} | | | | @|@{ignore first} = | Split To Lines | ${manylines} | | | | @|@{ignore last} = | Split To Lines | ${manylines} | | | | @|@{5th to 10th} = | Split To Lines | ${manylines} | | | | @|@{first two} = | Split To Lines | ${manylines} | | | | @|@{last two} = | Split To Lines | ${manylines} | | |

Use *Get Line* if you only need to get a single line.

**get_line** *(string, line_number)*

Returns the specified line from the given *string*.

Line numbering starts from 0 and it is possible to use negative indices to refer to lines from the end. The line is returned without the newline character.

Examples: | ${first} = | Get Line | ${string} | 0 | | | | ${2nd last} = | Get Line | ${string} | -2 |

**get_lines_containing_string** *(string, pattern, case_insensitive=False)*

Returns lines of the given *string* that contain the *pattern*.

The *pattern* is always considered to be a normal string and a line matches if the *pattern* is found anywhere in it. By default the match is case-sensitive, but setting *case_insensitive* to any value makes it case-insensitive.

Lines are returned as one string concatenated back together with newlines. Possible trailing newline is never returned. The number of matching lines is automatically logged.

Examples: | ${lines} = | Get Lines Containing String | ${result} | An example | | | ${ret} = | Get Lines Containing String | ${ret} | FAIL | case-insensitive |

See *Get Lines Matching Pattern* and *Get Lines Matching Regexp* if you need more complex pattern matching.

**get_lines_matching_pattern** *(string, pattern, case_insensitive=False)*

Returns lines of the given *string* that match the *pattern*.

The *pattern* is a _glob pattern_ where: | * | matches everything | | ? | matches any single character | | [chars] | matches any character inside square brackets (e.g. `[abc]` matches either ‘a’, ‘b’ or ‘c’) | | [!chars] | matches any character not inside square brackets | |

A line matches only if it matches the *pattern* fully. By default the match is case-sensitive, but setting *case_insensitive* to any value makes it case-insensitive.
Lines are returned as one string catenated back together with newlines. Possible trailing newline is never returned. The number of matching lines is automatically logged.

Examples:

```
${lines} = Get Lines Matching Pattern
${result} = Wild????
${ret} = Get Lines Matching Pattern
${ret} = FAIL: *
```

See *Get Lines Matching Regexp* if you need more complex patterns and *Get Lines Containing String* if searching literal strings is enough.

**get_lines_matching_regexp** (*string*, *pattern*)

Returns lines of the given *string* that match the regexp *pattern*.

See *BuiltIn.Should Match Regexp* for more information about Python regular expression syntax in general and how to use it in Robot Framework test data in particular. A line matches only if it matches the *pattern* fully. Notice that to make the match case-insensitive, you need to embed case-insensitive flag into the *pattern*.

Lines are returned as one string catenated back together with newlines. Possible trailing newline is never returned. The number of matching lines is automatically logged.

Examples:

```
${lines} = Get Lines Matching Regexp
${result} = Reg\w{3}
${ret} = (?i)FAIL: .*
```

See *Get Lines Matching Pattern* and *Get Lines Containing String* if you do not need full regular expression powers (and complexity).

**replace_string** (*string*, *search_for*, *replace_with*, *count=-1*)

Replaces *search_for* in the given *string* with *replace_with*. *search_for* is used as a literal string. See *Replace String Using Regexp* if more powerful pattern matching is needed.

If the optional argument *count* is given, only that many occurrences from left are replaced. Negative *count* means that all occurrences are replaced (default behaviour) and zero means that nothing is done.

A modified version of the string is returned and the original string is not altered.

Examples:

```
${str} = Replace String
${str} = Replace String Hi
```

**replace_string_using_regexp** (*string*, *pattern*, *replace_with*, *count=-1*)

Replaces *pattern* in the given *string* with *replace_with*. This keyword is otherwise identical to *_replace_string*, but the *pattern* to search for is considered to be a regular expression. See *BuiltIn.Should Match Regexp* for more information about Python regular expression syntax in general and how to use it in Robot Framework test data in particular.

Examples:

```
${str} = Replace String Using Regexp (Hello|Hi) Hei
```

**split_string** (*string*, *separator=None*, *max_split=-1*)

Splits the *string* using *separator* as a delimiter string.

If a *separator* is not given, any whitespace string is a separator. In that case also possible consecutive whitespace as well as leading and trailing whitespace is ignored.

Split words are returned as a list. If the optional *max_split* is given, at most *max_split* splits are done, and the returned list will have maximum *max_split + 1* elements.

Examples:

```
@{words} = Split String
@{words} = Split String
@{words} = Split String |
```

See *Split String From Right* if you want to start splitting from right, and *Fetch From Left* and *Fetch From Right* if you only want to get first/last part of the string.
**split_string_from_right**  
`(string, separator=None, max_split=-1)`  
Splits the `string` using `separator` starting from right.  

Same as `Split String`, but splitting is started from right. This has an effect only when `max_split` is given.  

Examples:  
| $(first) | $(others) = | Split String | $(string) | 1 | 1 | 1 | $(others) | $(last) = | Split String From Right | $(string) | 1 | 1 |

**split_string_to_characters**  
`(string)`  
Splits the string `*` to characters.  

Example:  
| @ | $(characters) = | Split String To Characters | $(string) |

**fetch_from_left**  
`(string, marker)`  
Returns contents of the `string` before the first occurrence of `marker`.  

If the `marker` is not found, whole string is returned.  

See also `Fetch From Right`, `Split String` and `Split String From Right`.  

**fetch_from_right**  
`(string, marker)`  
Returns contents of the `string` after the last occurrence of `marker`.  

If the `marker` is not found, whole string is returned.  

See also `Fetch From Left`, `Split String` and `Split String From Right`.  

**generate_random_string**  
`(length=8, chars=\[LETTERS\]|\[NUMBERS\])`  
Generates a string with a desired `length` from the given `chars`.  

The population sequence `chars` contains the characters to use when generating the random string. It can contain any characters, and it is possible to use special markers explained in the table below:

| \[LOWER\] | Lowercase ASCII characters from ‘a’ to ‘z’. |
| \[UPPER\] | Uppercase ASCII characters from ‘A’ to ‘Z’. |
| \[LETTERS\] | Lowercase and uppercase ASCII characters. |
| \[NUMBERS\] | Numbers from 0 to 9. |

Examples:  
| $(ret) = | Generate Random String | $(string) |
| $(low) = | Generate Random String | 12 | \[LOWER\] |
| $(bin) = | Generate Random String | 81011 | \[NUMBERS\]abcdef |

**get_substring**  
`(string, start, end=None)`  
Returns a substring from `start` index to `end` index.  

The `start` index is inclusive and `end` is exclusive. Indexing starts from 0, and it is possible to use negative indices to refer to characters from the end.  

Examples:  
| $(ignore first) = | Get Substring | $(string) |
| $(ignore last) = | Get Substring | $(string) |
| $(5th to 10th) = | Get Substring | $(string) |
| $(first two) = | Get Substring | $(string) |
| $(last two) = | Get Substring | $(string) |

**should_be_string**  
`(item, msg=None)`  
Fails if the given `item` is not a string.  

This keyword passes regardless is the `item` is a Unicode string or a byte string. Use `Should Be Unicode String` or `Should Be Byte String` if you want to restrict the string type.  

The default error message can be overridden with the optional `msg` argument.
should_not_be_string(item, msg=None)
Fails if the given item is a string.

The default error message can be overridden with the optional msg argument.

should_be_unicode_string(item, msg=None)
Fails if the given item is not a Unicode string.

Use Should Be Byte String if you want to verify the item is a byte string, or Should Be String if both Unicode and byte strings are fine.

The default error message can be overridden with the optional msg argument.

New in Robot Framework 2.7.7.

should_be_byte_string(item, msg=None)
Fails if the given item is not a byte string.

Use Should Be Unicode String if you want to verify the item is a Unicode string, or Should Be String if both Unicode and byte strings are fine.

The default error message can be overridden with the optional msg argument.

New in Robot Framework 2.7.7.

should_be_lowercase(string, msg=None)
Fails if the given string is not in lowercase.

For example ‘string’ and ‘with specials!’ would pass, and ‘String’,’’ and ‘’ would fail.

The default error message can be overridden with the optional msg argument.

See also Should Be Uppercase and Should Be Titlecase. All these keywords were added in Robot Framework 2.1.2.

should_be_uppercase(string, msg=None)
Fails if the given string is not in uppercase.

For example ‘STRING’ and ‘WITH SPECIALS!’ would pass, and ‘String’, ‘’ and ‘’ would fail.

The default error message can be overridden with the optional msg argument.

See also Should Be Titlecase and Should Be Lowercase. All these keywords were added in Robot Framework 2.1.2.

should_be_titlecase(string, msg=None)
Fails if given string is not title.

string is a titlecased string if there is at least one character in it, uppercase characters only follow uncased characters and lowercase characters only cased ones.

For example ‘This Is Title’ would pass, and ‘Word In UPPER’, ‘Word In lower’, ‘’ and ‘’ would fail.

The default error message can be overridden with the optional msg argument.

See also Should Be Uppercase and Should Be Lowercase. All theses keyword were added in Robot Framework 2.1.2.

Telnet Module

class robot.libraries.Telnet.Telnet (timeout='3 seconds', newline='CRLF', prompt=None, prompt_is_regexp=False, encoding='UTF-8', encoding_errors='ignore', default_log_level='INFO')
A test library providing communication over Telnet connections.
**Telnet** is Robot Framework’s standard library that makes it possible to connect to Telnet servers and execute commands on the opened connections.

== Table of contents ==

• Connections
• Writing and reading
• Configuration
• Importing
• Logging
• Time string format
• Shortcuts
• Keywords

= Connections =

The first step of using **Telnet** is opening a connection with **Open Connection** keyword. Typically the next step is logging in with **Login** keyword, and in the end the opened connection can be closed with **Close Connection**.

It is possible to open multiple connections and switch the active one using **Switch Connection**. **Close All Connections** can be used to close all the connections, which is especially useful in suite teardowns to guarantee that all connections are always closed.

= Writing and reading =

After opening a connection and possibly logging in, commands can be executed or text written to the connection for other reasons using **Write** and **Write Bare** keywords. The main difference between these two is that the former adds a [configured newline] after the text automatically.

After writing something to the connection, the resulting output can be read using **Read**, **Read Until**, **Read Until Regexp**, and **Read Until Prompt** keywords. Which one to use depends on the context, but the latest one is often the most convenient.

As a convenience when running a command, it is possible to use **Execute Command** that simply uses **Write** and **Read Until Prompt** internally. **Write Until Expected Output** is useful if you need to wait until writing something produces a desired output.

Written and read text is automatically encoded/decoded using a [configured encoding].

= Configuration =

Many aspects related the connections can be easily configured either globally or per connection basis. Global configuration is done when [library is imported], and these values can be overridden per connection by **Open Connection** or with setting specific keywords **Set Timeout**, **Set Newline**, **Set Prompt**, **Set Encoding**, and **Set Default Log Level**

== Timeout ==

Timeout defines how long is the maximum time to wait when reading output. It is used internally by **Read Until**, **Read Until Regexp**, **Read Until Prompt**, and **Login** keywords. The default value is 3 seconds.

== Newline ==

Newline defines which line separator **Write** keyword should use. The default value is **CRLF** that is typically used by Telnet connections.

Newline can be given either in escaped format using ‘n’ and ‘r’ or with special ‘LF’ and ‘CR’ syntax.

Examples: \Set Newline \n \| \Set Newline \CRLF \|
== Prompt ==

Often the easiest way to read the output of a command is reading all the output until the next prompt with Read Until Prompt. It also makes it easier, and faster, to verify did Login succeed.

Prompt can be specified either as a normal string or a regular expression. The latter is especially useful if the prompt changes as a result of the executed commands.

== Encoding ==

To ease handling text containing non-ASCII characters, all written text is encoded and read text decoded by default. The default encoding is UTF-8 that works also with ASCII. Encoding can be disabled by using a special encoding value NONE. This is mainly useful if you need to get the bytes received from the connection as-is.

Notice that when writing to the connection, only Unicode strings are encoded using the defined encoding. Byte strings are expected to be already encoded correctly. Notice also that normal text in test data is passed to the library as Unicode and you need to use variables to use bytes.

It is also possible to configure the error handler to use if encoding or decoding characters fails. Accepted values are the same that encode/decode functions in Python strings accept. In practice the following values are the most useful:

- ignore: ignore characters that cannot be encoded (default)
- strict: fail if characters cannot be encoded
- replace: replace characters that cannot be encoded with a replacement character

Examples: | Open Connection | lolcathost | encoding=Latin1 | encoding_errors=strict | Set Encoding | ISO-8859-15 | Set Encoding | errors=ignore |

Using UTF-8 encoding by default and being able to configure the encoding are new features in Robot Framework 2.7.6. In earlier versions only ASCII was supported and encoding errors were silently ignored. Robot Framework 2.7.7 added a possibility to specify the error handler, changed the default behavior back to ignoring encoding errors, and added the possibility to disable encoding.

== Default log level ==

Default log level specifies the log level keywords use for logging unless they are given an explicit log level. The default value is INFO, and changing it, for example, to DEBUG can be a good idea if there is lot of unnecessary output that makes log files big.

Configuring default log level in importing and with Open Connection are new features in Robot Framework 2.7.6. In earlier versions only Set Default Log Level could be used.

= Logging =

All keywords that read something log the output. These keywords take the log level to use as an optional argument, and if no log level is specified they use the [#Configuration|configured] default value.

The valid log levels to use are TRACE, DEBUG, INFO (default), and WARN. Levels below INFO are not shown in log files by default whereas warnings are shown more prominently.

The [http://docs.python.org/2/library/telnetlib.html|telnetlib module] used by this library has a custom logging system for logging content it sends and receives. Starting from Robot Framework 2.7.7, these low level log messages are forwarded to Robot’s log file using TRACE level.

= Time string format =

Timeouts and other times used must be given as a time string using format like ‘15 seconds’ or ‘1min 10s’. If the timeout is given as just a number, for example, ‘10’ or ‘1.5’, it is considered to be seconds. The time string format is described in more detail in an appendix of [http://code.google.com/p/robotframework/wiki/UserGuide|Robot Framework User Guide]].
Telnet library can be imported with optional configuration parameters.

Configuration parameters are used as default values when new connections are opened with Open Connection keyword. They can also be overridden after opening the connection using the Set Timeout, Set Newline, Set Prompt, Set Encoding, and Set Default Log Level keywords. See these keywords and Configuration section above for more information about these parameters and their possible values.

Examples (use only one of these):

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
<th>Value</th>
<th>Value</th>
<th>Value</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library</td>
<td>Telnet</td>
<td></td>
<td></td>
<td></td>
<td># default values</td>
</tr>
<tr>
<td>Library</td>
<td>Telnet</td>
<td>0.5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Library</td>
<td>Telnet</td>
<td>newline=LF</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Library</td>
<td>Telnet</td>
<td>newline=LF</td>
<td>encoding=ISO-8859-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Library</td>
<td>Telnet</td>
<td>2.0</td>
<td>LF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Library</td>
<td>Telnet</td>
<td>2.0</td>
<td>CRLF</td>
<td>$</td>
<td></td>
</tr>
<tr>
<td>Library</td>
<td>Telnet</td>
<td>2.0</td>
<td>LF</td>
<td>(&gt;</td>
<td># )</td>
</tr>
</tbody>
</table>

ROBOT_LIBRARY_SCOPE = ‘TEST_SUITE’

ROBOT_LIBRARY_VERSION = ‘2.7.7’

get_keyword_names()

open_connection (host, alias=None, port=23, timeout=None, newline=None, prompt=None, prompt_is_regexp=False, encoding=None, encoding_errors=None, default_log_level=None)

Opens a new Telnet connection to the given host and port.

The timeout, newline, prompt, prompt_is_regexp, encoding, and default_log_level arguments get default values when the library is [Importing] imported. Setting them here overrides those values for the opened connection. See Configuration section for more information.

Possible already opened connections are cached and it is possible to switch back to them using Switch Connection keyword. It is possible to switch either using explicitly given alias or using index returned by this keyword. Indexing starts from 1 and is reset back to it by Close All Connections keyword.

switch_connection (index_or_alias)

Switches between active connections using an index or an alias.

Aliases can be given to Open Connection keyword which also always returns the connection index.

This keyword returns the index of previous active connection.

Example: 1 Open Connection | myhost.net | | Login | john | secret | Write | some command | 2nd conn | Login | root | password | Write | another cmd | 3rd conn | | | | |

The example above expects that there were no other open connections when opening the first one, because it used index ‘1’ when switching to the connection later. If you are not sure about that, you can store the index into a variable as shown below.

${index} = | Open Connection | myhost.net |

Do Something
**Switch Connection** | ⋆{index} ⋆

**close_all_connections**()

Closes all open connections and empties the connection cache.

If multiple connections are opened, this keyword should be used in a test or suite teardown to make sure that all connections are closed. It is not an error if some of the connections have already been closed by *Close Connection*.

After this keyword, new indexes returned by *Open Connection* keyword are reset to 1.

class robot.libraries.Telnet.TelnetConnection(host=None, port=23, timeout=3.0, newline='CRLF', prompt=None, prompt_is-regexp=False, encoding='UTF-8', encoding_errors='ignore', default_log_level='INFO')

**Bases:** telnetlib.Telnet

**set_timeout** (timeout)

Sets the timeout used for waiting output in the current connection.

Read operations that expect some output to appear (*Read Until*, *Read Until Regexp*, *Read Until Prompt*, *Login*) use this timeout and fail if the expected output does not appear before this timeout expires.

The `timeout` must be given in *time string format*. The old timeout is returned and can be used to restore the timeout later.

Example: 1 | ${old} = | Set Timeout | 2 minute 30 seconds | 1 | Do Something | 1 | Set Timeout | 1 | ${old} | 1

See *Configuration* section for more information about global and connection specific configuration.

**set_newline** (newline)

Sets the newline used by *Write* keyword in the current connection.

The old newline is returned and can be used to restore the newline later. See *Set Timeout* for a similar example.

See *Configuration* section for more information about global and connection specific configuration.

**set_prompt** (prompt, prompt_is-regexp=False)

Sets the prompt used by *Read Until Prompt* and *Login* in the current connection.

If `prompt_is-regexp` is given any true value, including any non-empty string, the given `prompt` is considered to be a regular expression.

The old prompt is returned and can be used to restore the prompt later.

Example: 1 | ${prompt} | ${regexp} = | 1 Set Prompt | 1 Do Something | 1 | Set Prompt | 1 | ${prompt} | 1 | ${regexp} | 1

See the documentation of [http://docs.python.org/2/library/re.html#Python re module](http://docs.python.org/2/library/re.html#Python re module) for more information about the supported regular expression syntax. Notice that possible backslashes need to be escaped in Robot Framework test data.

See *Configuration* section for more information about global and connection specific configuration.

**set_encoding** (encoding=None, errors=None)

Sets the encoding to use for *writing and reading* in the current connection.

The given `encoding` specifies the encoding to use when written/read text is encoded/decoded, and `errors` specifies the error handler to use if encoding/decoding fails. Either of these can be omitted and in that case the old value is not affected. Use string *NONE* to disable encoding altogether.
See Configuration section for more information about encoding and error handlers, as well as global and connection specific configuration in general.

The old values are returned and can be used to restore the encoding and the error handler later. See Set Prompt for a similar example.

Setting encoding in general is a new feature in Robot Framework 2.7.6. Specifying the error handler and disabling encoding were added in 2.7.7.

**set_default_log_level**(level)
Sets the default log level used for logging in the current connection.

The old default log level is returned and can be used to restore the log level later.

See Configuration section for more information about global and connection specific configuration.

**close_connection**(loglevel=None)
Closes the current Telnet connection.

Remaining output in the connection is read, logged, and returned. It is not an error to close an already closed connection.

Use Close All Connections if you want to make sure all opened connections are closed.

See Logging section for more information about log levels.

**login**(username, password, login_prompt='login: ', password_prompt='Password: ', login_timeout='1 second', login_incorrect='Login incorrect')
Logs in to the Telnet server with the given user information.

This keyword reads from the connection until the login_prompt is encountered and then types the given username. Then it reads until the password_prompt and types the given password. In both cases a newline is appended automatically and the connection specific timeout used when waiting for outputs.

How logging status is verified depends on whether a prompt is set for this connection or not:

1) If the prompt is set, this keyword reads the output until the prompt is found using the normal timeout. If no prompt is found, login is considered failed and also this keyword fails. Note that in this case both login_timeout and login_incorrect arguments are ignored.

2) If the prompt is not set, this keyword sleeps until login_timeout and then reads all the output available on the connection. If the output contains login_incorrect text, login is considered failed and also this keyword fails. Both of these configuration parameters were added in Robot Framework 2.7.6. In earlier versions they were hard coded.

See Configuration section for more information about setting newline, timeout, and prompt.

**write**(text, loglevel=None)
Writes the given text plus a newline into the connection.

The newline character sequence to use can be [#Configuration|configured] both globally and per connection basis. The default value is CRLF.

This keyword consumes the written text, until the added newline, from the output and logs and returns it. The given text itself must not contain newlines. Use Write Bare instead if either of these features causes a problem.

*Note:* This keyword does not return the possible output of the executed command. To get the output, one of the Read ... keywords must be used. See Writing and reading section for more details.

See Logging section for more information about log levels.

**write_bare**(text)
Writes the given text, and nothing else, into the connection.
This keyword does not append a newline nor consume the written text. Use Write if these features are needed.

**write_until_expected_output** *(text, expected, timeout, retry_interval, loglevel=None)*

Writes the given text repeatedly, until expected appears in the output.

*text* is written without appending a newline and it is consumed from the output before trying to find *expected*. If *expected* does not appear in the output within *timeout*, this keyword fails.

*retry_interval* defines the time to wait *expected* to appear before writing the *text* again. Consuming the written *text* is subject to the normal [#Configuration|configured timeout].

Both *timeout* and *retry_interval* must be given in *time string format*. See Logging section for more information about log levels.

Example: | Write Until Expected Output | ps -ef | grep myprocess rn | myprocess | |
| ... | 5 s | 0.5 s |

The above example writes command `ps -ef | grep myprocess` until `myprocess` appears in the output. The command is written every 0.5 seconds and the keyword fails if `myprocess` does not appear in the output in 5 seconds.

**read** *(loglevel=None)*

Reads everything that is currently available in the output.

Read output is both returned and logged. See Logging section for more information about log levels.

**read_until** *(expected, loglevel=None)*

Reads output until *expected* text is encountered.

Text up to and including the match is returned and logged. If no match is found, this keyword fails. How much to wait for the output depends on the [#Configuration|configured timeout].

See Logging section for more information about log levels. Use Read Until Regexp if more complex matching is needed.

**read_until_regexp** *(expected)*

Reads output until any of the *expected* regular expressions match.

This keyword accepts any number of regular expressions patterns or compiled Python regular expression objects as arguments. Text up to and including the first match to any of the regular expressions is returned and logged. If no match is found, this keyword fails. How much to wait for the output depends on the [#Configuration|configured timeout].

If the last given argument is a [#Logging|valid log level], it is used as *loglevel* similarly as with Read Until keyword.

See the documentation of [http://docs.python.org/2/library/re.html|Python re module] for more information about the supported regular expression syntax. Notice that possible backslashes need to be escaped in Robot Framework test data.

Examples: | Read Until Regexp | (#|$) || Read Until Regexp | first_regexp | second_regexp || Read Until Regexp | \d{4}-\d{2}-\d{2} | DEBUG |

**read_until_prompt** *(loglevel=None)*

Reads output until the prompt is encountered.

This keyword requires the prompt to be [#Configuration|configured] either in importing or with Open Connection or Set Prompt keyword.

Text up to and including the prompt is returned and logged. If no prompt is found, this keyword fails. How much to wait for the output depends on the [#Configuration|configured timeout].

See Logging section for more information about log levels.
**execute_command** *(command, loglevel=None)*

Executes the given *command* and reads, logs, and returns everything until the prompt.

This keyword requires the prompt to be [#Configuration|configured] either in importing or with *Open Connection* or *Set Prompt* keyword.

This is a convenience keyword that uses *Write* and *Read Until Prompt* internally Following two examples are thus functionally identical:

```plaintext
$[out] = | Execute Command | pwd |
Write | pwd |
$[out] = | Read Until Prompt |
```

See *Logging* section for more information about log levels.

**msg** *(msg, *args)*

**close** *

Close the connection.

**expect** *(list, timeout=None)*

Read until one from a list of a regular expressions matches.

The first argument is a list of regular expressions, either compiled (re.RegexObject instances) or uncompiled (strings). The optional second argument is a timeout, in seconds; default is no timeout.

Return a tuple of three items: the index in the list of the first regular expression that matches; the match object returned; and the text read up till and including the match.

If EOF is read and no text was read, raise EOFError. Otherwise, when nothing matches, return (-1, None, text) where text is the text received so far (may be the empty string if a timeout happened).

If a regular expression ends with a greedy match (e.g. ’.*’) or if more than one expression can match the same input, the results are undeterministic, and may depend on the I/O timing.

**fileno** *

Return the fileno() of the socket object used internally.

**fill_rawq** *

Fill raw queue from exactly one recv() system call.

Block if no data is immediately available. Set self.eof when connection is closed.

**get_socket** *

Return the socket object used internally.

**interact** *

Interaction function, emulates a very dumb telnet client.

**listener** *

Helper for mt_interact() – this executes in the other thread.

**mt_interact** *

Multithreaded version of interact().

**open** *(host, port=0, timeout=<object object at 0x7f0990b40240>)*

Connect to a host.

The optional second argument is the port number, which defaults to the standard telnet port (23).
Don’t try to reopen an already connected instance.

**process_rawq()**
Transfer from raw queue to cooked queue.
Set self.eof when connection is closed. Don’t block unless in the midst of an IAC sequence.

**rawq_getchar()**
Get next char from raw queue.
Block if no data is immediately available. Raise EOFError when connection is closed.

**read_all()**
Read all data until EOF; block until connection closed.

**read_eager()**
Read readily available data.
Raise EOFError if connection closed and no cooked data available. Return ‘’ if no cooked data available otherwise. Don’t block unless in the midst of an IAC sequence.

**read_lazy()**
Process and return data that’s already in the queues (lazy).
Raise EOFError if connection closed and no data available. Return ‘’ if no cooked data available otherwise. Don’t block unless in the midst of an IAC sequence.

**read_sb_data()**
Return any data available in the SB ... SE queue.
Return ‘’ if no SB ... SE available. Should only be called after seeing a SB or SE command. When a new SB command is found, old unread SB data will be discarded. Don’t block.

**read_some()**
Read at least one byte of cooked data unless EOF is hit.
Return ‘’ if EOF is hit. Block if no data is immediately available.

**read_very_eager()**
Read everything that’s possible without blocking in I/O (eager).
Raise EOFError if connection closed and no cooked data available. Return ‘’ if no cooked data available otherwise. Don’t block unless in the midst of an IAC sequence.

**read_very_lazy()**
Return any data available in the cooked queue (very lazy).
Raise EOFError if connection closed and no data available. Return ‘’ if no cooked data available otherwise. Don’t block.

**set_debuglevel(debuglevel)**
Set the debug level.
The higher it is, the more debug output you get (on sys.stdout).

**set_option_negotiation_callback(callback)**
Provide a callback function called after each receipt of a telnet option.

**sock_avail()**
Test whether data is available on the socket.
XML Module

class robot.libraries.XML.XML
   Bases: object

Robot Framework test library for verifying and modifying XML documents.

As the name implies, XML is a test library for verifying contents of XML files. In practice it is a pretty thin wrapper on top of Python’s [http://docs.python.org/library/xml.etree.elementtree.html|ElementTree XML API].

The library has the following main usages:

• Parsing an XML file, or a string containing XML, into an XML element structure and finding certain elements from it for further analysis (e.g. Parse XML and Get Element keywords).
• Getting text or attributes of elements (e.g. Get Element Text and Get Element Attribute).
• Directly verifying text, attributes, or whole elements (e.g Element Text Should Be and Elements Should Be Equal).
• Modifying XML and saving it (e.g. Set Element Text, Add Element and Save XML).

== Table of contents ==

• Parsing XML
• Example
• Finding elements with xpath
• Element attributes
• Handling XML namespaces
• Shortcuts
• Keywords

= Parsing XML =

XML can be parsed into an element structure using Parse XML keyword. It accepts both paths to XML files and strings that contain XML. The keyword returns the root element of the structure, which then contains other elements as its children and their children.

The element structure returned by Parse XML, as well as elements returned by keywords such as Get Element, can be used as the source argument with other keywords. In addition to an already parsed XML structure, other keywords also accept paths to XML files and strings containing XML similarly as Parse XML.

When the source is given as a path to a file, the forward slash character (/) can be used as the path separator regardless the operating system. On Windows also the backslash works, but it the test data it needs to be escaped by doubling it (\). Using the built-in variable ${/} naturally works too.

= Example =

The following simple example demonstrates parsing XML and verifying its contents both using keywords in this library and in BuiltIn and Collections libraries. How to use xpath expressions to find elements and what attributes the returned elements contain are discussed, with more examples, in Finding elements with xpath and Element attributes sections.

In this example, as well as in many other examples in this documentation, ${XML} refers to the following example XML document. In practice ${XML} could either be a path to an XML file or it could contain the XML itself.

<example>
Notice that in the example three last lines are equivalent. Which one to use in practice depends on which other elements you need to get or verify. If you only need to do one verification, using the last line alone would suffice. If more verifications are needed, parsing the XML with *Parse XML* only once would be more efficient.

Finding elements with xpath =

ElementTree, and thus also this library, supports finding elements using xpath expressions. ElementTree does not, however, support the full xpath syntax, and what is supported depends on its version. ElementTree 1.3 that is distributed with Python and Jython 2.7 supports richer syntax than versions distributed with earlier Python interpreters.

The supported xpath syntax is explained below and [http://effbot.org/zone/element-xpath.htm|ElementTree documentation] provides more details. In the examples `$XML` refers to the same XML structure as in the earlier example.

== Tag names ==
When just a single tag name is used, xpath matches all direct child elements that have that tag name.

```robot
$root = | Parse XML | $XML | |
Should Be Equal | $root.tag | example | |
$first = | Get Element | $root | first | |
Should Be Equal | $first.text | text | |
Dictionary Should Contain Key | $first.attrib | id | |
Element Text Should Be | $first | text | |
Element Attribute Should Be | $first | id | 1 | |
Element Attribute Should Be | $root | id | 1 | xpath=first | |
Element Attribute Should Be | $XML | id | 1 | xpath=first | |
```
== Paths ==

Paths are created by combining tag names with a forward slash (/). For example, parent/child matches all child elements under parent element. Notice that if there are multiple parent elements that all have child elements, parent/child xpath will match all these child elements.

```robot
${elem} = | Get Element | ${XML} | second/child |
Should Be Equal | ${elem.tag} | child |
${elem} = | Get Element | ${XML} | third/child/grandchild |
Should Be Equal | ${elem.tag} | grandchild |
```

== Wildcards ==

An asterisk (*) can be used in paths instead of a tag name to denote any element.

```robot
@{children} = | Get Elements | ${XML} | */child |
Length Should Be | ${children} | 3 |
```

== Current element ==

The current element is denoted with a dot (.) Normally the current element is implicit and does not need to be included in the xpath.

== Parent element ==

The parent element of another element is denoted with two dots (..). Notice that it is not possible to refer to the parent of the current element. This syntax is supported only in ElementTree 1.3 (i.e. Python/Jython 2.7 and newer).

```robot
${elem} = | Get Element | ${XML} | */second/.. |
Should Be Equal | ${elem.tag} | third |
```

== Search all sub elements ==

Two forward slashes (//) mean that all sub elements, not only the direct children, are searched. If the search is started from the current element, an explicit dot is required.

```robot
@{elements} = | Get Elements | ${XML} | ./second |
Length Should Be | ${elements} | 2 |
${b} = | Get Element | ${XML} | html/b |
Should Be Equal | ${b.text} | bold |
```

== Predicates ==

Predicates allow selecting elements using also other criteria than tag names, for example, attributes or position. They are specified after the normal tag name or path using syntax `path[predicate]`. The path can have wildcards and other special syntax explained above.
What predicates ElementTree supports is explained in the table below. Notice that predicates in general are supported only in ElementTree 1.3 (i.e. Python/Jython 2.7 and newer).

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Matches</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>@attrib</td>
<td>Elements with attribute attrib</td>
<td>second[@id]</td>
</tr>
<tr>
<td>@attrib=&quot;value&quot;</td>
<td>Elements with attribute attrib having value value</td>
<td>*[@id=&quot;2&quot;]</td>
</tr>
<tr>
<td>position</td>
<td>Elements at the specified position. Position can be an integer (starting from 1), expression last(), or relative expression like last() - 1</td>
<td>third/child[1]</td>
</tr>
<tr>
<td>tag</td>
<td>Elements with a child element named tag.</td>
<td>third/child[grandchild]</td>
</tr>
</tbody>
</table>

Predicates can also be stacked like path[predicate1][predicate2]. A limitation is that possible position predicate must always be first.

= Element attributes =

All keywords returning elements, such as Parse XML, and Get Element, return ElementTree’s [http://docs.python.org/library/xml.etree.elementtree.html#xml.etree.ElementTree.ElementTree.Element|Element objects]. These elements can be used as inputs for other keywords, but they also contain several useful attributes that can be accessed directly using the extended variable syntax.

The attributes that are both useful and convenient to use in the test data are explained below. Also other attributes, including methods, can be accessed, but that is typically better to do in custom libraries than directly in the test data.

The examples use the same $/XML/ structure as the earlier examples.

== tag ==

The tag of the element.

```
${root} = | Parse XML | ${XML} |
Should Be Equal | ${root.tag} | example |
```

== text ==

The text that the element contains or Python None if the element has no text. Notice that the text _does not_ contain texts of possible child elements nor text after or between children. Notice also that in XML whitespace is significant, so the text contains also possible indentation and newlines. To get also text of the possible children, optionally whitespace normalized, use Get Element Text keyword.

```
${1st} = | Get Element | ${XML} |
Should Be Equal | ${1st.text} | text |
${2nd} = | Get Element | ${XML} |
Should Be Equal | ${2nd.text} | ${NONE} |
${p} = | Get Element | ${XML} |
Should Be Equal | ${p.text} | n${SPACE*6}Text with${SPACE} |
```

== tail ==
The text after the element before the next opening or closing tag. Python None if the element has no tail. Similarly as with text, also tail contains possible indentation and newlines.

```
${b} = | Get Element | ${XML} | html/p/b |
Should Be Equal | ${b.tail} | ${SPACE}and${SPACE} |
```

== attrib ==

A Python dictionary containing attributes of the element.

```
${2nd} = | Get Element | ${XML} | second |
Should Be Equal | ${2nd.attrib['id']} | 2 |
${3rd} = | Get Element | ${XML} | third |
Should Be Empty | ${3rd.attrib} |
```

= Handling XML namespaces =

ElementTree handles possible namespaces in XML documents by adding the namespace URI to tag names in so-called Clark Notation. That is inconvenient especially with xpaths, and by default this library strips those namespaces away and moves them to xmlns attribute instead. That can be avoided by passing keep_clark_notation argument to Parse XML keyword. The pros and cons of both approaches are discussed in more detail below.

== How ElementTree handles namespaces ==

If an XML document has namespaces, ElementTree adds namespace information to tag names in [http://www.jclark.com/xml/xmlns.htm|Clark Notation] (e.g. `{http://ns.uri}tag` and removes original xmlns attributes. This is done both with default namespaces and with namespaces with a prefix. How it works in practice is illustrated by the following example, where `$({NS})` variable contains this XML document:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns="http://www.w3.org/1999/xhtml">
    <xsl:template match="/">
        <html></html>
    </xsl:template>
</xsl:stylesheet>
```

```
$root = | Parse XML | $({NS}) | keep_clark_notation=yes |
Should Be Equal | $root.tag | {http://www.w3.org/1999/XSL/Transform}stylesheet |
Element Should Exist | $root |
{http://www.w3.org/1999/XSL/Transform}template/{http://www.w3.org/1999/xhtml}html |
Should Be Empty | $root.attrib |
```

As you can see, including the namespace URI in tag names makes xpaths really long and complex.

If you save the XML, ElementTree moves namespace information back to xmlns attributes. Unfortunately it does not restore the original prefixes:
The resulting output is semantically same as the original, but mangling prefixes like this may still not be desirable. Notice also that the actual output depends slightly on ElementTree version.

== Default namespace handling ==

Because the way ElementTree handles namespaces makes xpaths so complicated, this library, by default, strips namespaces from tag names and moves that information back to xmlns attributes. How this works in practice is shown by the example below, where $/NS$ variable contains the same XML document as in the previous example.

${root} = | Parse XML | $/NS | 
Should Be Equal | ${root.tag} | stylesheet |
Element Should Exist | ${root} | template/html |
Element Attribute Should Be | ${root} | xmlns | http://www.w3.org/1999/XSL/Transform |
Element Attribute Should Be | ${root} | xmlns | http://www.w3.org/1999/xhtml | xpath=template/html |

Now that tags do not contain namespace information, xpaths are simple again.

A minor limitation of this approach is that namespace prefixes are lost. As a result the saved output is not exactly same as the original one in this case either:

<stylesheet xmlns="http://www.w3.org/1999/XSL/Transform">
  <template match="/">
    <html xmlns="http://www.w3.org/1999/xhtml"/>
  </template>
</stylesheet>

Also this output is semantically same as the original. If the original XML had only default namespaces, the output would also looks identical.

== Attribute namespaces ==

Attributes in XML documents are, by default, in the same namespaces as the element they belong to. It is possible to use different namespaces by using prefixes, but this is pretty rare.

If an attribute has a namespace prefix, ElementTree will replace it with Clark Notation the same way it handles elements. Because stripping namespaces from attributes could cause attribute conflicts, this library does not handle attribute namespaces at all. Thus the following example works the same way regardless how namespaces are handled.

${root} = | Parse XML | <root id="1" ns:id="2" xmlns:ns="http://my.ns"/> |
Element Attribute Should Be | ${root} | id | 1 |
Element Attribute Should Be | ${root} | {http://my.ns}id | 2 |
parse_xml(source, keep_clark_notation=False)

Parses the given XML file or string into an element structure. The source can either be a path to an XML file or a string containing XML. In both cases the XML is parsed into ElementTree [http://docs.python.org/library/xml.etree.elementtree.html#xml.etree.ElementTree.ElementTree] and the root element is returned.

As discussed in Handling XML namespaces section, this keyword, by default, strips possible namespaces added by ElementTree into tag names. This typically eases handling XML documents with namespaces considerably. If you do not want that to happen, or want to avoid the small overhead of going through the element structure when your XML does not have namespaces, you can disable this feature by giving keep_clark_notation argument a true value (e.g. any non-empty string).

Examples:
| ${root} = Parse XML | <root><child/></root> |
| ${xml} = Parse XML | ${CUR-DIR}/test.xml | no namespace cleanup |

Use Get Element keyword if you want to get a certain element and not the whole structure. See Parsing XML section for more details and examples

Stripping namespaces is a new feature in Robot Framework 2.7.5.

get_element(source, xpath='.')

Returns an element in the source matching the xpath.

The source can be a path to an XML file, a string containing XML, or an already parsed XML element. The xpath specifies which element to find. See the introduction for more details about both the possible sources and the supported xpath syntax.

The keyword fails if more, or less, than one element matches the xpath. Use Get Elements if you want all matching elements to be returned.

Examples using ${XML} structure from the introduction:
| $[root] = | Parse XML | <root><child/></root> | 1 | $[xml] = | Parse XML | ${CUR-DIR}/test.xml | no namespace cleanup |

Use Get Element keyword if you want to get a certain element and not the whole structure. See Parsing XML section for more details and examples

Stripping namespaces is a new feature in Robot Framework 2.7.5.

get_elements(source, xpath)

Returns a list of elements in the source matching the xpath.

The source can be a path to an XML file, a string containing XML, or an already parsed XML element. The xpath specifies which element to find. See the introduction for more details.

Elements matching the xpath are returned as a list. If no elements match, an empty list is returned. Use Get Element if you want to get exactly one match.

Examples using ${XML} structure from the introduction:

Parse XML is recommended for parsing XML when the whole structure is needed. It must be used if there is a need to configure how XML namespaces are handled.

get_child_elements(source, xpath='.')

Returns the child elements of the specified element as a list.

The element whose children to return is specified using source and xpath. They have exactly the same semantics as with Get Element keyword.

All the direct child elements of the specified element are returned. If the element has no children, an empty list is returned.
Examples using `$XML$` structure from the *introduction*:

```
| $\text{children} = $ Get Child Elements $\text{XML}$ |
| Length Should Be $\text{children}$ |
| $\text{children} = $ Get Child Elements $\text{XML}$ |
| xpath=first |
| Should Be Empty $\text{children}$ |
```

**get_element_count** (*source, xpath='.'*)

Returns and logs how many elements the given `xpath` matches.

Arguments `source` and `xpath` have exactly the same semantics as with `Get Elements` keyword that this keyword uses internally.

See also `Element Should Exist` and `Element Should Not Exist`.

New in Robot Framework 2.7.5.

**element_should_exist** (*source, xpath='.', message=None*)

Verifies that one or more element match the given `xpath`.

Arguments `source` and `xpath` have exactly the same semantics as with `Get Elements` keyword. Keyword passes if the `xpath` matches one or more elements in the `source`. The default error message can be overridden with the `message` argument.

See also `Element Should Not Exist` as well as `Get Element Count` that this keyword uses internally.

New in Robot Framework 2.7.5.

**element_should_not_exist** (*source, xpath='.', message=None*)

Verifies that no element match the given `xpath`.

Arguments `source` and `xpath` have exactly the same semantics as with `Get Elements` keyword. Keyword fails if the `xpath` matches any element in the `source`. The default error message can be overridden with the `message` argument.

See also `Element Should Exist` as well as `Get Element Count` that this keyword uses internally.

New in Robot Framework 2.7.5.

**get_element_text** (*source, xpath='.', normalize_whitespace=False*)

Returns all text of the element, possibly whitespace normalized.

The element whose text to return is specified using `source` and `xpath`. They have exactly the same semantics as with `Get Element` keyword.

This keyword returns all the text of the specified element, including all the text its children and grandchildren contains. If the element has no text, an empty string is returned. As discussed in the *introduction*, the returned text is thus not always the same as the `text` attribute of the element.

Be default all whitespace, including newlines and indentation, inside the element is returned as-is. If `normalize_whitespace` is given any true value (e.g. any non-empty string), then leading and trailing whitespace is stripped, newlines and tabs converted to spaces, and multiple spaces collapsed into one. This is especially useful when dealing with HTML data.

Examples using `$XML$` structure from the *introduction*:

```
| $\text{text} = $ Get Element Text $\text{XML}$ |
| Should Be Equal $\text{text}$ |
| Should Be Equal $\text{text}$ |
| Should Be Empty $\text{text}$ |
| Should Be Empty $\text{text}$ |
| $\text{paragraph} = $ Get Element $\text{XML}$ |
| $\text{paragraph} = $ Get Element $\text{XML}$ |
| $\text{paragraph} = $ Get Element $\text{XML}$ |
```

See also `Get Elements Texts`, `Element Text Should Be` and `Element Text Should Match`.

**get_elements_texts** (*source, xpath, normalize_whitespace=False*)

Returns text of all elements matching `xpath` as a list.

The elements whose text to return is specified using `source` and `xpath`. They have exactly the same semantics as with `Get Elements` keyword.
The text of the matched elements is returned using the same logic as with Get Element Text. This includes optional whitespace normalization using the normalize_whitespace option.

Examples using $\{XML\}$ structure from the introduction:

```
| @{texts} = | Get Elements Texts | $\{XML\}$ |
|        | third/child | Length Should Be | $\{texts\}$ |
|        | [0] | Should Be Equal | $\{texts\}[1] |

element_text_should_be (source, expected, xpath='.', normalize_whitespace=False, message=None)

Verifies that the text of the specified element is expected.

The element whose text is verified is specified using source and xpath. They have exactly the same semantics as with Get Element keyword.

The text to verify is got from the specified element using the same logic as with Get Element Text. This includes optional whitespace normalization using the normalize_whitespace option.

The keyword passes if the text of the element is equal to the expected value, and otherwise it fails. The default error message can be overridden with the message argument. Use Element Text Should Match to verify the text against a pattern instead of an exact value.

Examples using $\{XML\}$ structure from the introduction:

```
| Element Text Should Be | $\{XML\}$ |
| text |

get_element_attribute (source, name, xpath='.', default=None)

Returns the named attribute of the specified element.

The element whose attribute to return is specified using source and xpath. They have exactly the same semantics as with Get Element keyword.

The value of the attribute name of the specified element is returned. If the element does not have such element, the default value is returned instead.

Examples using $\{XML\}$ structure from the introduction:

```
| $\{attribute\} = | Get Element Attribute | $\{XML\}$ |
| id | xpath=first | Should Be Equal | $\{attribute\}$ |

get_element_attributes (source, xpath='.')

Returns all attributes of the specified element.

The element whose attributes to return is specified using source and xpath. They have exactly the same semantics as with Get Element keyword.
Attributes are returned as a Python dictionary. It is a copy of the original attributes so modifying it has no effect on the XML structure.

Examples using $\{XML\}$ structure from the introduction:

```
| ${attributes} = | Get Element Attributes |
| $\{XML\} | first | Dictionary Should Contain Key | $\{attributes\} | id | $\{attributes\} = | Get Element Attributes | $\{XML\} | third | Should Be Empty | $\{attributes\} |
```

Use Get Element Attribute to get the value of a single attribute.

**element_attribute_should_be** *(source, name, expected, xpath='.', message=None)*  
Verifies that the specified attribute is *expected*.  
The element whose attribute is verified is specified using *source* and *xpath*. They have exactly the same semantics as with *Get Element* keyword.  
The keyword passes if the attribute *name* of the element is equal to the *expected* value, and otherwise it fails. The default error message can be overridden with the *message* argument.  
To test that the element does not have a certain attribute, Python *None* (i.e. variable $\{NONE\}$) can be used as the *expected* value. A cleaner alternative is using *Element Should Not Have Attribute*.  
Examples using $\{XML\}$ structure from the introduction:

```
| Element Attribute Should Be | $\{XML\} | id | 1 |
| Element Attribute Should Be | $\{XML\} | id | $\{NONE\} |
```

See also *Element Attribute Should Match* and *Get Element Attribute*.

**element_attribute_should_match** *(source, name, pattern, xpath='.', message=None)*  
Verifies that the specified attributes matches *expected*.  
This keyword works exactly like *Element Attribute Should Be* except that the expected value can be given as a pattern that the attribute of the element must match.  
Pattern matching is similar as matching files in a shell, and it is always case-sensitive. In the pattern, '*' matches anything and '?' matches any single character.  
Examples using $\{XML\}$ structure from the introduction:

```
| Element Attribute Should Match | $\{XML\} | id | 1 |
| Element Attribute Should Match | $\{XML\} | id | c*d |
```

**element_should_not_have_attribute** *(source, name, xpath='.', message=None)*  
Verifies that the specified element does not have attribute *name*.  
The element whose attribute is verified is specified using *source* and *xpath*. They have exactly the same semantics as with *Get Element* keyword.  
The keyword fails if the specified element has attribute *name*. The default error message can be overridden with the *message* argument.  
Examples using $\{XML\}$ structure from the introduction:

```
| Element Should Not Have Attribute | $\{XML\} | id | 1 |
| Element Should Not Have Attribute | $\{XML\} | xxx |
```

See also *Get Element Attribute*, *Get Element Attributes*, *Element Text Should Be* and *Element Text Should Match*.

New in Robot Framework 2.7.5.

**elements_should_be_equal** *(source, expected, exclude_children=False, normalize_whitespace=False)*  
Verifies that the given *source* element is equal to *expected*.  
Both *source* and *expected* can be given as a path to an XML file, as a string containing XML, or as an already parsed XML element structure. See *introduction* for more information about parsing XML in general.
The keyword passes if the *source* element and *expected* element are equal. This includes testing the tag names, texts, and attributes of the elements. By default also child elements are verified the same way, but this can be disabled by setting `exclude_children` to any true value (e.g. any non-empty string).

All texts inside the given elements are verified, but possible text outside them is not. By default texts must match exactly, but setting `normalize_whitespace` to any true value makes text verification independent on newlines, tabs, and the amount of spaces. For more details about handling text see `Get Element Text` keyword and discussion about elements’ text and tail attributes in the *introduction*.

Examples using `$XML` structure from the *introduction*:

```
| ${first} | Get Element | ${XML} | first |
| Elements Should Be Equal | ${first} | <first id=?">text</first> |
| html/p | Elements Should Be Equal | ${p} | <p>Text with <b>bold</b> and <i>italics</i>.</p> |
| normalize_whitespace=yes | Elements Should Be Equal | ${p} | <p>Text with</p> |
```

The last example may look a bit strange because the `<p>` element only has text *Text with*. The reason is that rest of the text inside `<p>` actually belongs to the child elements.

See also *Elements Should Match*.

**elements_should_match** *(source, expected, exclude_children=False, normalize_whitespace=False)*

Verifies that the given `source` element matches `expected`.

This keyword works exactly like *Elements Should Be Equal* except that texts and attribute values in the expected value can be given as patterns.

Pattern matching is similar as matching files in a shell, and it is always case-sensitive. In the pattern, ‘*’ matches anything and ‘?’ matches any single character.

Examples using `$XML` structure from the *introduction*:

```
| $(first) | Get Element | $(XML) | first |
| Elements Should Match | $(first) | <first id=""">text</first> |
```

See *Elements Should Be Equal* for more examples.

**set_element_tag** *(source, tag, xpath='.')*

Sets the tag of the specified element to `tag`.

The element whose tag to set is specified using `source` and `xpath`. They have exactly the same semantics as with `Get Element` keyword. The given `source` structure is modified and also returned.

Examples using `$XML` structure from *Example*:

```
```

New in Robot Framework 2.7.5.

**set_element_text** *(source, text=None, tail=None, xpath='.')*

Sets text and/or tail text of the specified element.

The element whose text to set is specified using `source` and `xpath`. They have exactly the same semantics as with `Get Element` keyword. The given `source` structure is modified and also returned.

Element’s text and tail text are changed only if new `text` and/or `tail` values are given. See *Element attributes* section for more information about text and tail in general.

Examples using `$XML` structure from *Example*:

```
| Set Element Text | $XML | new text | xpath=first |
```

New in Robot Framework 2.7.5.
set_element_attribute (source, name, value, xpath='.')

Sets attribute name of the specified element to value.

The element whose attribute to set is specified using source and xpath. They have exactly the same semantics as with Get Element keyword. The given source structure is modified and also returned.

It is possible to both set new attributes and to overwrite existing. Use Remove Element Attribute or Remove Element Attributes for removing them.

Examples using $<XML> structure from Example: | Set Element Attribute | $<XML> | attr | value || Element Attribute Should Be | $<XML> | id | new | xpath=first || Element Attribute Should Be | $<XML> | id | new | xpath=first |

New in Robot Framework 2.7.5.

remove_element_attribute (source, name, xpath='.')

Removes attribute name from the specified element.

The element whose attribute to remove is specified using source and xpath. They have exactly the same semantics as with Get Element keyword. The given source structure is modified and also returned.

It is not a failure to remove a non-existing attribute. Use Remove Element Attributes to remove all attributes and Set Element Attribute to set them.

Examples using $<XML> structure from Example: | Remove Element Attribute | $<XML> | id | xpath=first || Element Should Not Have Attribute | $<XML> | id | xpath=first |

New in Robot Framework 2.7.5.

remove_element_attributes (source, xpath='.')

Removes all attributes from the specified element.

The element whose attributes to remove is specified using source and xpath. They have exactly the same semantics as with Get Element keyword. The given source structure is modified and also returned.

Use Remove Element Attribute to remove a single attribute and Set Element Attribute to set them.

Examples using $<XML> structure from Example: | Remove Element Attributes | $<XML> | xpath=first || Element Should Not Have Attribute | $<XML> | id | xpath=first |

New in Robot Framework 2.7.5.

add_element (source, element, index=None, xpath='.')

Adds a child element to the specified element.

The element to whom to add the new element is specified using source and xpath. They have exactly the same semantics as with Get Element keyword. The given source structure is modified and also returned.

The element to add can be specified as a path to an XML file or as a string containing XML, or it can be an already parsed XML element. The element is copied before adding so modifying either the original or the added element has no effect on the other. The element is added as the last child by default, but a custom index can be used to alter the position. Indices start from zero (0 = first position, 1 = second position, etc.), and negative numbers refer to positions at the end (-1 = second last position, -2 = third last, etc.).

Examples using $<XML> structure from Example: | Add Element | $<XML> | <new id="x"><c1/></new> || Add Element | $<XML> | <c2/> | xpath=new || Add Element | $<XML> | <c3/> | index=1 | xpath=new || $<new> = | Get Element | $<XML> | new || Elements Should Be Equal | $<new> | <new id="x">=<c1/>=<<c3/>=<<c2/></new> |

Use Remove Element or Remove Elements to remove elements.

New in Robot Framework 2.7.5.
**remove_element** *(source, xpath='', remove_tail=False)*

Removes the element matching `xpath` from the `source` structure.

The element to remove from the `source` is specified with `xpath` using the same semantics as with `Get Element` keyword. The given `source` structure is modified and also returned.

The keyword fails if `xpath` does not match exactly one element. Use `Remove Elements` to remove all matched elements.

Element’s tail text is not removed by default, but that can be changed by giving `remove_tail` a true value (e.g. any non-empty string). See Element attributes section for more information about tail in general.

Examples using `${XML}` structure from Example:

| Remove Element | ${XML} | xpath=second | \| Element Should Not Exist | ${XML} | xpath=second | \| Element Should Not Exist | ${XML} | xpath=html/p/b | remove_tail=yes | \| Element Text Should Be | ${XML} | Text with italics. | \| xpath=html/p | normalize_whitespace=yes |

New in Robot Framework 2.7.5.

**remove_elements** *(source, xpath='', remove_tail=False)*

Removes all elements matching `xpath` from the `source` structure.

The elements to remove from the `source` are specified with `xpath` using the same semantics as with `Get Elements` keyword. The given `source` structure is modified and also returned.

It is not a failure if `xpath` matches no elements. Use `Remove Element` to remove exactly one element and `Add Element` to add new ones.

Element’s tail text is not removed by default, but that can be changed by using `remove_tail` argument similarly as with `Remove Element`.

Examples using `${XML}` structure from Example:

| Remove Elements | ${XML} | xpath=*/child | \| Element Should Not Exist | ${XML} | xpath=second/child | \| Element Should Not Exist | ${XML} | xpath=third/child |

New in Robot Framework 2.7.5.

**clear_element** *(source, xpath='', clear_tail=False)*

Clears the contents of the specified element.

The element to clear is specified using `source` and `xpath`. They have exactly the same semantics as with `Get Element` keyword. The given `source` structure is modified and also returned.

Clearing the element means removing its text, attributes, and children. Element’s tail text is not removed by default, but that can be changed by giving `clear_tail` a true value (e.g. any non-empty string). See Element attributes section for more information about tail in general.

Examples using `${XML}` structure from Example:

| Clear Element | ${XML} | xpath=first | \| Elements Should Be Equal | ${first} = | \| Get Element | ${XML} | xpath=first | \| Elements Should Be Equal | ${first} | <first/> | \| Clear Element | ${XML} | xpath=html/p/b | clear_tail=yes | \| Element Text Should Be | ${XML} | Text with italics. | \| xpath=html/p | normalize_whitespace=yes | \| Clear Element | ${XML} | Elements Should Be Equal | ${XML} | <example/> |

Use `Remove Element` to remove the whole element.

New in Robot Framework 2.7.5.

**copy_element** *(source, xpath='.')*

Returns a copy of the specified element.

The element to copy is specified using `source` and `xpath`. They have exactly the same semantics as with `Get Element` keyword.

If the copy or the original element is modified afterwards, the changes have no effect on the other.
Examples using `${XML}` structure from Example:  

```robot
| ${elem} = Get Element | ${XML} | xpath=first |
| ${copy1} = Copy Element | ${elem} | ${copy2} = Copy Element | ${XML} | xpath=first | Set Element Text | ${XML} | new text | xpath=first | Set Element Attribute | ${copy1} | id | new | Elements Should Be Equal | ${elem} | <first id="1">new text</first> | Elements Should Be Equal | ${copy1} | <first id="new">text</first> | Elements Should Be Equal | ${copy2} | <first id="1">text</first> |
```

New in Robot Framework 2.7.5.

**element_to_string**(source, xpath='.')

Returns the string representation of the specified element.

The element to convert to a string is specified using `source` and `xpath`. They have exactly the same semantics as with `Get Element` keyword.

The returned string is in Unicode format and it does not contain any XML declaration.

See also `Log Element` and `Save XML`.

**log_element**(source, level='INFO', xpath='.')

Logs the string representation of the specified element.

The element specified with `source` and `xpath` is first converted into a string using `Element To String` keyword internally. The resulting string is then logged using the given `level`.

The logged string is also returned.

**save_xml**(source, path, encoding='UTF-8')

Saves the given element to the specified file.

The element to save is specified with `source` using the same semantics as with `Get Element` keyword.

The file where the element is saved is denoted with `path` and the encoding to use with `encoding`. The resulting file contains an XML declaration.

Use `Element To String` if you just need a string representation of the element,

New in Robot Framework 2.7.5.

**class** robot.libraries.XML.ElementComparator(comparator, normalizer=None, exclude_children=False)

**class** robot.libraries.XML.Location(path, is_root=True)

**class** robot.libraries.XML.Location(path, is_root=True)

**class** robot.libraries.dialogs_ipy.MessageDialog(message)

**class** robot.libraries.dialogs_ipy.InputDialog(message, default)

**class** robot.libraries.dialogs_ipy.SelectionDialog(message, options)

**class** robot.libraries.dialogs_ipy.PassFailDialog(message)
dialogs_jy Module

dialogs_py Module

class robot.libraries.dialogs_py.MessageDialog (message, value=None)
    Bases: robot.libraries.dialogs_py._TkDialog

    after (ms, func=None, *args)
        Call function once after given time.
        MS specifies the time in milliseconds. FUNC gives the function which shall be called. Additional parameters are given as parameters to the function call. Return identifier to cancel scheduling with after_cancel.

    after_cancel (id)
        Cancel scheduling of function identified with ID.
        Identifier returned by after or after_idle must be given as first parameter.

    after_idle (func, *args)
        Call FUNC once if the Tcl main loop has no event to process.
        Return an identifier to cancel the scheduling with after_cancel.

    aspect (minNumer=None, minDenom=None, maxNumer=None, maxDenom=None)
        Instruct the window manager to set the aspect ratio (width/height) of this widget to be between MINNUMER/MINDENOM and MAXNUMER/MAXDENOM. Return a tuple of the actual values if no argument is given.

    attributes (*args)
        This subcommand returns or sets platform specific attributes
        The first form returns a list of the platform specific flags and their values. The second form returns the value for the specific option. The third form sets one or more of the values. The values are as follows:
        On Windows, -disabled gets or sets whether the window is in a disabled state. -toolwindow gets or sets the style of the window to toolwindow (as defined in the MSDN). -topmost gets or sets whether this is a topmost window (displays above all other windows).
        On Macintosh, XXXXX
        On Unix, there are currently no special attribute values.

    bbox (column=None, row=None, col2=None, row2=None)
        Return a tuple of integer coordinates for the bounding box of this widget controlled by the geometry manager grid.
        If COLUMN, ROW is given the bounding box applies from the cell with row and column 0 to the specified cell. If COL2 and ROW2 are given the bounding box starts at that cell.
        The returned integers specify the offset of the upper left corner in the master widget and the width and height.

    bell (displayof=0)
        Ring a display’s bell.

    bind (sequence=None, func=None, add=None)
        Bind to this widget at event SEQUENCE a call to function FUNC.
        SEQUENCE is a string of concatenated event patterns. An event pattern is of the form <MODIFIER-MODIFIER-TYPE-DETAIL> where MODIFIER is one of Control, Mod2, M2, Shift, Mod3, M3, Lock, Mod4, M4, Button1, B1, Mod5, M5 Button2, B2, Meta, M, Button3, B3, Alt, Button4, B4, Double,
Button5, B5 Triple, Mod1, M1. TYPE is one of Activate, Enter, Map, Button1, Button, Expose, Motion, ButtonRelease FocusIn, MouseWheel, Circulate, FocusOut, Property, Colormap, Gravity Reparent, Configure, KeyPress, Key, Unmap, Deactivate, KeyRelease Visibility, Destroy, Leave and DETAIL is the button number for ButtonPress, ButtonRelease and DETAIL is the Keysym for KeyPress and KeyRelease. Examples are <Control-Button-1> for pressing Control and mouse button 1 or <Alt-A> for pressing A and the Alt key (KeyPress can be omitted). An event pattern can also be a virtual event of the form <<AString>> where AString can be arbitrary. This event can be generated by event_generate. If events are concatenated they must appear shortly after each other.

FUNC will be called if the event sequence occurs with an instance of Event as argument. If the return value of FUNC is “break” no further bound function is invoked.

An additional boolean parameter ADD specifies whether FUNC will be called additionally to the other bound function or whether it will replace the previous function.

Bind will return an identifier to allow deletion of the bound function with unbind without memory leak.

If FUNC or SEQUENCE is omitted the bound function or list of bound events are returned.

```
bind_all (sequence=None, func=None, add=None)
```

Bind to all widgets at an event SEQUENCE a call to function FUNC. An additional boolean parameter ADD specifies whether FUNC will be called additionally to the other bound function or whether it will replace the previous function. See bind for the return value.

```
bind_class (className, sequence=None, func=None, add=None)
```

Bind to widgets with bindtag CLASSNAME at event SEQUENCE a call of function FUNC. An additional boolean parameter ADD specifies whether FUNC will be called additionally to the other bound function or whether it will replace the previous function. See bind for the return value.

```
bindtags (tagList=None)
```

Set or get the list of bindtags for this widget.

With no argument return the list of all bindtags associated with this widget. With a list of strings as argument the bindtags are set to this list. The bindtags determine in which order events are processed (see bind).

```
cget (key)
```

Return the resource value for a KEY given as string.

```
client (name=None)
```

Store NAME in WM_CLIENT_MACHINE property of this widget. Return current value.

```
clipboard_append (string, **kw)
```

Append STRING to the Tk clipboard.

A widget specified at the optional displayof keyword argument specifies the target display. The clipboard can be retrieved with selection_get.

```
clipboard_clear (**kw)
```

Clear the data in the Tk clipboard.

A widget specified for the optional displayof keyword argument specifies the target display.

```
clipboard_get (**kw)
```

Retrieve data from the clipboard on window’s display.

The window keyword defaults to the root window of the Tkinter application.

The type keyword specifies the form in which the data is to be returned and should be an atom name such as STRING or FILE_NAME. Type defaults to STRING, except on X11, where the default is to try UTF8_STRING and fall back to STRING.

This command is equivalent to:
colormapwindows (*wlist)
Store list of window names (WLIST) into WM_COLORMAPWINDOWS property of this widget. This
list contains windows whose colormaps differ from their parents. Return current list of widgets if WLIST
is empty.

colormodel (value=None)
Useless. Not implemented in Tk.

columnconfigure (index, cnf={}, **kw)
Configure column INDEX of a grid.

    Valid resources are minsize (minimum size of the column), weight (how much does additional space
    propagate to this column) and pad (how much space to let additionally).

command (value=None)
Store VALUE in WM_COMMAND property. It is the command which shall be used to invoke the appli-
cation. Return current command if VALUE is None.

config (cnf=None, **kw)
Configure resources of a widget.

    The values for resources are specified as keyword arguments. To get an overview about the allowed
    keyword arguments call the method keys.

configure (cnf=None, **kw)
Configure resources of a widget.

    The values for resources are specified as keyword arguments. To get an overview about the allowed
    keyword arguments call the method keys.

deciconify ()
Deiconify this widget. If it was never mapped it will not be mapped. On Windows it will raise this widget
and give it the focus.

deletecommand (name)
Internal function.

    Delete the Tcl command provided in NAME.

destroy ()
Destroy this and all descendants widgets.

event_add (virtual, *sequences)
Bind a virtual event VIRTUAL (of the form <<Name>>) to an event SEQUENCE such that the virtual
event is triggered whenever SEQUENCE occurs.

event_delete (virtual, *sequences)
Unbind a virtual event VIRTUAL from SEQUENCE.

event_generate (sequence, **kw)
Generate an event SEQUENCE. Additional keyword arguments specify parameter of the event (e.g. x, y,
rootx, rooty).

event_info (virtual=None)
Return a list of all virtual events or the information about the SEQUENCE bound to the virtual event
VIRTUAL.

focus ()
Direct input focus to this widget.
If the application currently does not have the focus this widget will get the focus if the application gets the focus through the window manager.

**focus_displayof()**
Return the widget which has currently the focus on the display where this widget is located.
Return None if the application does not have the focus.

**focus_force()**
Direct input focus to this widget even if the application does not have the focus. Use with caution!

**focus_get()**
Return the widget which has currently the focus in the application.
Use focus_displayof to allow working with several displays. Return None if application does not have the focus.

**focus_lastfor()**
Return the widget which would have the focus if top level for this widget gets the focus from the window manager.

**focus_set()**
Direct input focus to this widget.
If the application currently does not have the focus this widget will get the focus if the application gets the focus through the window manager.

**focusmodel(model=None)**
Set focus model to MODEL. “active” means that this widget will claim the focus itself, “passive” means that the window manager shall give the focus. Return current focus model if MODEL is None.

**frame()**
Return identifier for decorative frame of this widget if present.

**geometry(newGeometry=None)**
Set geometry to NEWGEOMETRY of the form =widthxheight+x+y. Return current value if None is given.

**getboolean(s)**
Return a boolean value for Tcl boolean values true and false given as parameter.

**getdouble**
alias of float

**getint**
alias of int

**getvar(name='PY_VAR')**
Return value of Tcl variable NAME.

**grab_current()**
Return widget which has currently the grab in this application or None.

**grab_release()**
Release grab for this widget if currently set.

**grab_set()**
Set grab for this widget.
A grab directs all events to this and descendant widgets in the application.

**grab_set_global()**
Set global grab for this widget.
A global grab directs all events to this and descendant widgets on the display. Use with caution - other applications do not get events anymore.

**grab_status()**

Return None, “local” or “global” if this widget has no, a local or a global grab.

**grid(baseWidth=None, baseHeight=None, widthInc=None, heightInc=None)**

Instruct the window manager that this widget shall only be resized on grid boundaries. WIDTHINC and HEIGHTINC are the width and height of a grid unit in pixels. BASEWIDTH and BASEHEIGHT are the number of grid units requested in Tk_GeometryRequest.

**grid_bbox(column=None, row=None, col2=None, row2=None)**

Return a tuple of integer coordinates for the bounding box of this widget controlled by the geometry manager grid.

If COLUMN, ROW is given the bounding box applies from the cell with row and column 0 to the specified cell. If COL2 and ROW2 are given the bounding box starts at that cell.

The returned integers specify the offset of the upper left corner in the master widget and the width and height.

**grid_columnconfigure(index, cnf={}, **kw)**

Configure column INDEX of a grid.

Valid resources are minsize (minimum size of the column), weight (how much does additional space propagate to this column) and pad (how much space to let additionally).

**grid_location(x, y)**

Return a tuple of column and row which identify the cell at which the pixel at position X and Y inside the master widget is located.

**grid_propagate(flag=['_noarg_'])**

Set or get the status for propagation of geometry information.

A boolean argument specifies whether the geometry information of the slaves will determine the size of this widget. If no argument is given, the current setting will be returned.

**grid_rowconfigure(index, cnf={}, **kw)**

Configure row INDEX of a grid.

Valid resources are minsize (minimum size of the row), weight (how much does additional space propagate to this row) and pad (how much space to let additionally).

**grid_size()**

Return a tuple of the number of column and rows in the grid.

**grid_slaves(row=None, column=None)**

Return a list of all slaves of this widget in its packing order.

**group(pathName=None)**

Set the group leader widgets for related widgets to PATHNAME. Return the group leader of this widget if None is given.

**iconbitmap(bitmap=None, default=None)**

Set bitmap for the iconified widget to BITMAP. Return the bitmap if None is given.

Under Windows, the DEFAULT parameter can be used to set the icon for the widget and any descendants that don’t have an icon set explicitly. DEFAULT can be the relative path to a .ico file (example: root.iconbitmap(default='myicon.ico')). See Tk documentation for more information.

**iconify()**

Display widget as icon.
iconmask (bitmap=None)
    Set mask for the icon bitmap of this widget. Return the mask if None is given.

iconname (newName=None)
    Set the name of the icon for this widget. Return the name if None is given.

iconposition (x=None, y=None)
    Set the position of the icon of this widget to X and Y. Return a tuple of the current values of X and X if
    None is given.

iconwindow (pathName=None)
    Set widget PATHNAME to be displayed instead of icon. Return the current value if None is given.

image_names ()
    Return a list of all existing image names.

image_types ()
    Return a list of all available image types (e.g. photo bitmap).

keys ()
    Return a list of all resource names of this widget.

lift (aboveThis=None)
    Raise this widget in the stacking order.

lower (belowThis=None)
    Lower this widget in the stacking order.

mainloop (n=0)
    Call the mainloop of Tk.

maxsize (width=None, height=None)
    Set max WIDTH and HEIGHT for this widget. If the window is gridded the values are given in grid units.
    Return the current values if None is given.

minsize (width=None, height=None)
    Set min WIDTH and HEIGHT for this widget. If the window is gridded the values are given in grid units.
    Return the current values if None is given.

nametowidget (name)
    Return the Tkinter instance of a widget identified by its Tcl name NAME.

option_add (pattern, value, priority=None)
    Set a VALUE (second parameter) for an option PATTERN (first parameter).
    An optional third parameter gives the numeric priority (defaults to 80).

option_clear ()
    Clear the option database.
    It will be reloaded if option_add is called.

option_get (name, className)
    Return the value for an option NAME for this widget with CLASSNAME.
    Values with higher priority override lower values.

option_readfile (fileName, priority=None)
    Read file FILENAME into the option database.
    An optional second parameter gives the numeric priority.
overridedirect (boolean=None)
Instruct the window manager to ignore this widget if BOOLEAN is given with 1. Return the current value if None is given.

pack_propagate (flag=['_noarg_'])
Set or get the status for propagation of geometry information.
A boolean argument specifies whether the geometry information of the slaves will determine the size of this widget. If no argument is given the current setting will be returned.

pack_slaves ()
Return a list of all slaves of this widget in its packing order.

place_slaves ()
Return a list of all slaves of this widget in its packing order.

positionfrom (who=None)
Instruct the window manager that the position of this widget shall be defined by the user if WHO is “user”, and by its own policy if WHO is “program”.

propagate (flag=['_noarg_'])
Set or get the status for propagation of geometry information.
A boolean argument specifies whether the geometry information of the slaves will determine the size of this widget. If no argument is given the current setting will be returned.

protocol (name=None, func=None)
Bind function FUNC to command NAME for this widget. Return the function bound to NAME if None is given. NAME could be e.g. “WM_SAVE_YOURSELF” or “WM_DELETE_WINDOW”.

quit ()
Quit the Tcl interpreter. All widgets will be destroyed.

register (func, subst=None, needcleanup=1)
Return a newly created Tcl function. If this function is called, the Python function FUNC will be executed. An optional function SUBST can be given which will be executed before FUNC.

resizable (width=None, height=None)
Instruct the window manager whether this width can be resized in WIDTH or HEIGHT. Both values are boolean values.

rowconfigure (index, cnf={}, **kw)
Configure row INDEX of a grid.
Valid resources are minsize (minimum size of the row), weight (how much does additional space propagate to this row) and pad (how much space to let additionally).

selection_clear (**kw)
Clear the current X selection.

selection_get (**kw)
Return the contents of the current X selection.
A keyword parameter selection specifies the name of the selection and defaults to PRIMARY. A keyword parameter displayof specifies a widget on the display to use. A keyword parameter type specifies the form of data to be fetched, defaulting to STRING except on X11, where UTF8_STRING is tried before STRING.

selection_handle (command, **kw)
Specify a function COMMAND to call if the X selection owned by this widget is queried by another application.
This function must return the contents of the selection. The function will be called with the arguments OFFSET and LENGTH which allows the chunking of very long selections. The following keyword parameters can be provided: selection - name of the selection (default PRIMARY), type - type of the selection (e.g. STRING, FILE_NAME).

\texttt{selection\_own (**kw)}

Become owner of X selection.

A keyword parameter selection specifies the name of the selection (default PRIMARY).

\texttt{selection\_own\_get (**kw)}

Return owner of X selection.

The following keyword parameter can be provided: selection - name of the selection (default PRIMARY), type - type of the selection (e.g. STRING, FILE_NAME).

\texttt{send (interp, cmd, *args)}

Send Tcl command CMD to different interpreter INTERP to be executed.

\texttt{setvar (name='PY\_VAR', value='1')}  
Set Tcl variable NAME to VALUE.

\texttt{show ()}  

\texttt{size ()}  
Return a tuple of the number of column and rows in the grid.

\texttt{sizefrom (who=None)}  
Instruct the window manager that the size of this widget shall be defined by the user if WHO is “user”, and by its own policy if WHO is “program”.

\texttt{slaves ()}  
Return a list of all slaves of this widget in its packing order.

\texttt{state (newstate=None)}  
Query or set the state of this widget as one of normal, icon, iconic (see \texttt{wm\_iconwindow}), withdrawn, or zoomed (Windows only).

\texttt{title (string=None)}  
Set the title of this widget.

\texttt{tk\_bisque ()}  
Change the color scheme to light brown as used in Tk 3.6 and before.

\texttt{tk\_focusFollowsMouse ()}  
The widget under mouse will get automatically focus. Can not be disabled easily.

\texttt{tk\_focusNext ()}  
Return the next widget in the focus order which follows widget which has currently the focus.

The focus order first goes to the next child, then to the children of the child recursively and then to the next sibling which is higher in the stacking order. A widget is omitted if it has the takefocus resource set to 0.

\texttt{tk\_focusPrev ()}  
Return previous widget in the focus order. See \texttt{tk\_focusNext} for details.

\texttt{tk\_menuBar (*args)}  
Do not use. Needed in Tk 3.6 and earlier.

\texttt{tk\_setPalette (*args, **kw)}  
Set a new color scheme for all widget elements.

A single color as argument will cause that all colors of Tk widget elements are derived from this. Alternatively several keyword parameters and its associated colors can be given. The following keywords are
valid: activeBackground, foreground, selectColor, activeForeground, highlightBackground, selectBackground, background, highlightColor, selectForeground, disabledForeground, insertBackground, troughColor.

**tk_strictMotif**(boolean=None)
Set Tcl internal variable, whether the look and feel should adhere to Motif.
A parameter of 1 means adhere to Motif (e.g. no color change if mouse passes over slider). Returns the set value.

**tkraise**(aboveThis=None)
Raise this widget in the stacking order.

**transient**(master=None)
Instruct the window manager that this widget is transient with regard to widget MASTER.

**unbind**(sequence, funcid=None)
Unbind for this widget for event SEQUENCE the function identified with FUNCID.

**unbind_all**(sequence)
Unbind for all widgets for event SEQUENCE all functions.

**unbind_class**(className, sequence)
Unbind for a all widgets with bindtag CLASSNAME for event SEQUENCE all functions.

**update**()
Enter event loop until all pending events have been processed by Tcl.

**update_idletasks**()
Enter event loop until all idle callbacks have been called. This will update the display of windows but not process events caused by the user.

**wait_variable**(name='PY_VAR')
Wait until the variable is modified.
A parameter of type IntVar, StringVar, DoubleVar or BooleanVar must be given.

**wait_visibility**(window=None)
Wait until the visibility of a WIDGET changes (e.g. it appears).
If no parameter is given self is used.

**wait_window**(window=None)
Wait until a WIDGET is destroyed.
If no parameter is given self is used.

**waitvar**(name='PY_VAR')
Wait until the variable is modified.
A parameter of type IntVar, StringVar, DoubleVar or BooleanVar must be given.

**winfo_atom**(name, displayof=0)
Return integer which represents atom NAME.

**winfo_atomname**(id, displayof=0)
Return name of atom with identifier ID.

**winfo_cells**()
Return number of cells in the colormap for this widget.

**winfo_children**()
Return a list of all widgets which are children of this widget.

---

4.1. robot Package
Robot Framework Documentation, Release 2.7.7

winfo_class()
   Return window class name of this widget.

winfo_colormapfull()
   Return true if at the last color request the colormap was full.

winfo_containing(rootX, rootY, displayof=0)
   Return the widget which is at the root coordinates ROOTX, ROOTY.

winfo_depth()
   Return the number of bits per pixel.

winfo_exists()
   Return true if this widget exists.

winfo_fpixels(number)
   Return the number of pixels for the given distance NUMBER (e.g. “3c”) as float.

winfo_geometry()
   Return geometry string for this widget in the form “widthxheight+X+Y”.

winfo_height()
   Return height of this widget.

winfo_id()
   Return identifier ID for this widget.

winfo_interps(displayof=0)
   Return the name of all Tcl interpreters for this display.

winfo_ismapped()
   Return true if this widget is mapped.

winfo_manager()
   Return the window mananger name for this widget.

winfo_name()
   Return the name of this widget.

winfo_parent()
   Return the name of the parent of this widget.

winfo_pathname(id, displayof=0)
   Return the pathname of the widget given by ID.

winfo_pixels(number)
   Rounded integer value of winfo_fpixels.

winfo_pointerx()
   Return the x coordinate of the pointer on the root window.

winfo_pointery()
   Return a tuple of x and y coordinates of the pointer on the root window.

winfo_pointery()
   Return the y coordinate of the pointer on the root window.

winfo_reqheight()
   Return requested height of this widget.

winfo_reqwidth()
   Return requested width of this widget.
\textbf{winfo\_rgb}(color)

Return tuple of decimal values for red, green, blue for COLOR in this widget.

\textbf{winfo\_rootx}()

Return x coordinate of upper left corner of this widget on the root window.

\textbf{winfo\_rooty}()

Return y coordinate of upper left corner of this widget on the root window.

\textbf{winfo\_screen}()

Return the screen name of this widget.

\textbf{winfo\_screencells}()

Return the number of the cells in the colormap of the screen of this widget.

\textbf{winfo\_screendepth}()

Return the number of bits per pixel of the root window of the screen of this widget.

\textbf{winfo\_screenheight}()

Return the number of pixels of the height of the screen of this widget in pixel.

\textbf{winfo\_screenmmheight}()

Return the number of pixels of the height of the screen of this widget in mm.

\textbf{winfo\_screenwidth}()

Return the number of pixels of the width of the screen of this widget in pixel.

\textbf{winfo\_screenvisual}()

Return one of the strings directcolor, grayscale, pseudocolor, staticcolor, staticgray, or truecolor for the
default colormodel of this screen.

\textbf{winfo\_screenwidth}()

Return the number of pixels of the width of the screen of this widget in pixel.

\textbf{winfo\_server}()

Return information of the X-Server of the screen of this widget in the form “XmajorRminor vendor ven-
dorVersion”.

\textbf{winfo\_toplevel}()

Return the toplevel widget of this widget.

\textbf{winfo\_viewable}()

Return true if the widget and all its higher ancestors are mapped.

\textbf{winfo\_visual}()

Return one of the strings directcolor, grayscale, pseudocolor, staticcolor, staticgray, or truecolor for the
colormodel of this widget.

\textbf{winfo\_visualid}()

Return the X identifier for the visual for this widget.

\textbf{winfo\_visualsavailable}(includeids=0)

Return a list of all visuals available for the screen of this widget.

Each item in the list consists of a visual name (see \texttt{winfo\_visual}), a depth and if INCLUDEIDS=1 is given
also the X identifier.

\textbf{winfo\_vrootheight}()

Return the height of the virtual root window associated with this widget in pixels. If there is no virtual root
window return the height of the screen.
winfo_vrootwidth()
Return the width of the virtual root window associated with this widget in pixel. If there is no virtual root
window return the width of the screen.

winfo_vrootx()
Return the x offset of the virtual root relative to the root window of the screen of this widget.

winfo_vrooty()
Return the y offset of the virtual root relative to the root window of the screen of this widget.

winfo_width()
Return the width of this widget.

winfo_x()
Return the x coordinate of the upper left corner of this widget in the parent.

winfo_y()
Return the y coordinate of the upper left corner of this widget in the parent.

withdraw()
Withdraw this widget from the screen such that it is unmapped and forgotten by the window manager.
Re-draw it with wm_deiconify.

wm_aspect(minNumer=None, minDenom=None, maxNumer=None, maxDenom=None)
Instruct the window manager to set the aspect ratio (width/height) of this widget to be between MINNU-
MER/MINDENOM and MAXNUMER/MAXDENOM. Return a tuple of the actual values if no argument
is given.

wm_attributes(*args)
This subcommand returns or sets platform specific attributes
The first form returns a list of the platform specific flags and their values. The second form returns the
value for the specific option. The third form sets one or more of the values. The values are as follows:
On Windows, -disabled gets or sets whether the window is in a disabled state. -toolwindow gets or sets
the style of the window to toolwindow (as defined in the MSDN). -topmost gets or sets whether this is a
topmost window (displays above all other windows).
On Macintosh, XXXXX
On Unix, there are currently no special attribute values.

wm_client(name=None)
Store NAME in WM_CLIENT_MACHINE property of this widget. Return current value.

wm_colormapwindows(*wlist)
Store list of window names (WLIST) into WM_COLORMAPWINDOWS property of this widget. This
list contains windows whose colormaps differ from their parents. Return current list of widgets if WLIST
is empty.

wm_command(value=None)
Store VALUE in WM_COMMAND property. It is the command which shall be used to invoke the appli-
cation. Return current command if VALUE is None.

wm_deiconify()
Deiconify this widget. If it was never mapped it will not be mapped. On Windows it will raise this widget
and give it the focus.

wm_focusmodel(model=None)
Set focus model to MODEL. “active” means that this widget will claim the focus itself, “passive” means
that the window manager shall give the focus. Return current focus model if MODEL is None.
wm_frame()
   Return identifier for decorative frame of this widget if present.

wm_geometry(newGeometry=None)
   Set geometry to NEWGEOMETRY of the form =widthxheight+x+y. Return current value if None is given.

wm_grid(baseWidth=None, baseHeight=None, widthInc=None, heightInc=None)
   Instruct the window manager that this widget shall only be resized on grid boundaries. WIDTHINC and
   HEIGHTINC are the width and height of a grid unit in pixels. BASEWIDTH and BASEHEIGHT are the
   number of grid units requested in Tk_GeometryRequest.

wm_group(pathName=None)
   Set the group leader widgets for related widgets to PATHNAME. Return the group leader of this widget if
   None is given.

wm_iconbitmap(bitmap=None, default=None)
   Set bitmap for the iconified widget to BITMAP. Return the bitmap if None is given.

   Under Windows, the DEFAULT parameter can be used to set the icon for the widget and any descen-
   dents that don’t have an icon set explicitly. DEFAULT can be the relative path to a .ico file (example:
   root.iconbitmap(default='myicon.ico') ). See Tk documentation for more information.

wm_iconify()
   Display widget as icon.

wm_iconmask(bitmap=None)
   Set mask for the icon bitmap of this widget. Return the mask if None is given.

wm_iconname(newName=None)
   Set the name of the icon for this widget. Return the name if None is given.

wm_iconposition(x=None, y=None)
   Set the position of the icon of this widget to X and Y. Return a tuple of the current values of X and Y if
   None is given.

wm_iconwindow(pathName=None)
   Set widget PATHNAME to be displayed instead of icon. Return the current value if None is given.

wm_maxsize(width=None, height=None)
   Set max WIDTH and HEIGHT for this widget. If the window is gridded the values are given in grid units.
   Return the current values if None is given.

wm_minsize(width=None, height=None)
   Set min WIDTH and HEIGHT for this widget. If the window is gridded the values are given in grid units.
   Return the current values if None is given.

wm_overrideredirect(boolean=None)
   Instruct the window manager to ignore this widget if BOOLEAN is given with 1. Return the current value
   if None is given.

wm_positionfrom(who=None)
   Instruct the window manager that the position of this widget shall be defined by the user if WHO is “user”,
   and by its own policy if WHO is “program”.

wm_protocol(name=None, func=None)
   Bind function FUNC to command NAME for this widget. Return the function bound to NAME if None is
   given. NAME could be e.g. “WM_SAVE_YOURSELF” or “WM_DELETE_WINDOW”.

wm_resizable(width=None, height=None)
   Instruct the window manager whether this width can be resized in WIDTH or HEIGHT. Both values are
   boolean values.
\texttt{wm\_sizefrom}(\texttt{who=None})

Instruct the window manager that the size of this widget shall be defined by the user if WHO is “user”, and by its own policy if WHO is “program”.

\texttt{wm\_state}(\texttt{newstate=None})

Query or set the state of this widget as one of normal, icon, iconic (see \texttt{wm\_iconwindow}), withdrawn, or zoomed (Windows only).

\texttt{wm\_title}(\texttt{string=None})

Set the title of this widget.

\texttt{wm\_transient}(\texttt{master=None})

Instruct the window manager that this widget is transient with regard to widget MASTER.

\texttt{wm\_withdraw}()

Withdraw this widget from the screen such that it is unmapped and forgotten by the window manager. Re-draw it with \texttt{wm\_deiconify}.

\texttt{class robot.libraries.dialogs\_py.InputDialog}(\texttt{message, default=''}\)

\texttt{Bases: robot.libraries.dialogs\_py._TkDialog}

\texttt{after}(\texttt{ms, func=None, *args})

Call function once after given time.

MS specifies the time in milliseconds. FUNC gives the function which shall be called. Additional parameters are given as parameters to the function call. Return identifier to cancel scheduling with \texttt{after\_cancel}.

\texttt{after\_cancel}(\texttt{id})

Cancel scheduling of function identified with ID.

Identifier returned by after or after_idle must be given as first parameter.

\texttt{after\_idle}(\texttt{func, *args})

Call FUNC once if the Tcl main loop has no event to process.

Return an identifier to cancel the scheduling with \texttt{after\_cancel}.

\texttt{aspect}(\texttt{minNumer=None, minDenom=None, maxNumer=None, maxDenom=None})

Instruct the window manager to set the aspect ratio (width/height) of this widget to be between MINNU-MER/MINDENOM and MAXNUMBER/MAXDENOM. Return a tuple of the actual values if no argument is given.

\texttt{attributes}(\texttt{*args})

This subcommand returns or sets platform specific attributes

The first form returns a list of the platform specific flags and their values. The second form returns the value for the specific option. The third form sets one or more of the values. The values are as follows:

On Windows, -disabled gets or sets whether the window is in a disabled state. -toolwindow gets or sets the style of the window to toolwindow (as defined in the MSDN). -topmost gets or sets whether this is a topmost window (displays above all other windows).

On Macintosh, XXXXX

On Unix, there are currently no special attribute values.

\texttt{bbox}(\texttt{column=None, row=None, col2=None, row2=None})

Return a tuple of integer coordinates for the bounding box of this widget controlled by the geometry manager grid.

If COLUMN, ROW is given the bounding box applies from the cell with row and column 0 to the specified cell. If COL2 and ROW2 are given the bounding box starts at that cell.
The returned integers specify the offset of the upper left corner in the master widget and the width and height.

**bell** *(displayof=0)*

Ring a display’s bell.

**bind** *(sequence=None, func=None, add=None)*

Bind to this widget at event `SEQUENCE` a call to function `FUNC`.

`SEQUENCE` is a string of concatenated event patterns. An event pattern is of the form `<MODIFIER-MODIFIER-TYPE-DETAIL>` where `MODIFIER` is one of Control, Mod2, M2, Shift, Mod3, M3, Lock, Mod4, M4, Button1, B1, Mod5, M5 Button2, B2, Meta, M, Button3, B3, Alt, Button4, B4, Double, Button5, B5 Triple, Mod1, M1. `TYPE` is one of Activate, Enter, Map, ButtonPress, Button, Expose, Motion, ButtonRelease FocusIn, MouseWheel, Circulate, FocusOut, Property, Colormap, Gravity Reparent, Configure, KeyPress, Key, Unmap, Deactivate, KeyRelease Visibility, Destroy, Leave and `DETAIL` is the button number for ButtonPress, ButtonRelease and `DETAIL` is the Keysym for KeyPress and KeyRelease. Examples are `<Control-Button-1>` for pressing Control and mouse button 1 or `<Alt-A>` for pressing A and the Alt key (KeyPress can be omitted). An event pattern can also be a virtual event of the form `<<AString>>` where `AString` can be arbitrary. This event can be generated by `event_generate`. If events are concatenated they must appear shortly after each other.

`FUNC` will be called if the event sequence occurs with an instance of `Event` as argument. If the return value of `FUNC` is “break” no further bound function is invoked.

An additional boolean parameter `ADD` specifies whether `FUNC` will be called additionally to the other bound function or whether it will replace the previous function.

Bind will return an identifier to allow deletion of the bound function with `unbind` without memory leak.

If `FUNC` or `SEQUENCE` is omitted the bound function or list of bound events are returned.

**bind_all** *(sequence=None, func=None, add=None)*

Bind to all widgets at an event `SEQUENCE` a call to function `FUNC`. An additional boolean parameter `ADD` specifies whether `FUNC` will be called additionally to the other bound function or whether it will replace the previous function. See `bind` for the return value.

**bind_class** *(className, sequence=None, func=None, add=None)*

Bind to widgets with `bindtag` `CLASSNAME` at event `SEQUENCE` a call of function `FUNC`. An additional boolean parameter `ADD` specifies whether `FUNC` will be called additionally to the other bound function or whether it will replace the previous function. See `bind` for the return value.

**bindtags** *(tagList=None)*

Set or get the list of bindtags for this widget.

With no argument return the list of all bindtags associated with this widget. With a list of strings as argument the bindtags are set to this list. The bindtags determine in which order events are processed (see `bind`).

**cget** *(key)*

Return the resource value for a `KEY` given as string.

**client** *(name=None)*

Store `NAME` in `WM_CLIENT_MACHINE` property of this widget. Return current value.

**clipboard_append** *(string, **kw)*

Append `STRING` to the Tk clipboard.

A widget specified at the optional `displayof` keyword argument specifies the target display. The clipboard can be retrieved with `selection_get`.

**clipboard_clear** *(**kw)*

Clear the data in the Tk clipboard.
A widget specified for the optional display of keyword argument specifies the target display.

**clipboard_get(** *kw**)

Retrieve data from the clipboard on window’s display.

The window keyword defaults to the root window of the Tkinter application.

The type keyword specifies the form in which the data is to be returned and should be an atom name such as STRING or FILE_NAME. Type defaults to STRING, except on X11, where the default is to try UTF8_STRING and fall back to STRING.

This command is equivalent to:

```
selection_get(CLIPBOARD)
```

**colormapwindows(** *wlist**)

Store list of window names (WLIST) into WM_COLORMAPWINDOWS property of this widget. This list contains windows whose colormaps differ from their parents. Return current list of widgets if WLIST is empty.

**colormodel** *(value=None)*

Useless. Not implemented in Tk.

**columnconfigure** *(index, cnf={}, **kw**)*

Configure column INDEX of a grid.

Valid resources are minsize (minimum size of the column), weight (how much does additional space propagate to this column) and pad (how much space to let additionally).

**command** *(value=None)*

Store VALUE in WM_COMMAND property. It is the command which shall be used to invoke the application. Return current command if VALUE is None.

**config** *(cnf=None, **kw**)*

Configure resources of a widget.

The values for resources are specified as keyword arguments. To get an overview about the allowed keyword arguments call the method keys.

**configure** *(cnf=None, **kw**)*

Configure resources of a widget.

The values for resources are specified as keyword arguments. To get an overview about the allowed keyword arguments call the method keys.

**deiconify** ()

Deiconify this widget. If it was never mapped it will not be mapped. On Windows it will raise this widget and give it the focus.

**deletecommand** *(name)*

Internal function.

Delete the Tcl command provided in NAME.

**destroy** ()

Destroy this and all descendants widgets.

**event_add** *(virtual, *sequences)*

Bind a virtual event VIRTUAL (of the form <<Name>>) to an event SEQUENCE such that the virtual event is triggered whenever SEQUENCE occurs.

**event_delete** *(virtual, *sequences)*

Unbind a virtual event VIRTUAL from SEQUENCE.
event_generate(sequence, **kw)
   Generate an event SEQUENCE. Additional keyword arguments specify parameter of the event (e.g. x, y, rootx, rooty).

event_info(virtual=None)
   Return a list of all virtual events or the information about the SEQUENCE bound to the virtual event VIRTUAL.

focus()
   Direct input focus to this widget.
   If the application currently does not have the focus this widget will get the focus if the application gets the focus through the window manager.

focus_displayof()
   Return the widget which has currently the focus on the display where this widget is located.
   Return None if the application does not have the focus.

focus_force()
   Direct input focus to this widget even if the application does not have the focus. Use with caution!

focus_get()
   Return the widget which has currently the focus in the application.
   Use focus_displayof to allow working with several displays. Return None if application does not have the focus.

focus_lastfor()
   Return the widget which would have the focus if top level for this widget gets the focus from the window manager.

focus_set()
   Direct input focus to this widget.
   If the application currently does not have the focus this widget will get the focus if the application gets the focus through the window manager.

focusmodel(model=None)
   Set focus model to MODEL. "active" means that this widget will claim the focus itself, "passive" means that the window manager shall give the focus. Return current focus model if MODEL is None.

frame()
   Return identifier for decorative frame of this widget if present.

geometry(newGeometry=None)
   Set geometry to NEWGEOMETRY of the form =widthxheight+x+y. Return current value if None is given.

getboolean(s)
   Return a boolean value for Tcl boolean values true and false given as parameter.

getdouble
   alias of float

getint
   alias of int

getvar(name='PY_VAR')
   Return value of Tcl variable NAME.

grab_current()
   Return widget which has currently the grab in this application or None.
grab_release()
Release grab for this widget if currently set.

grab_set()
Set grab for this widget.
A grab directs all events to this and descendant widgets in the application.

grab_set_global()
Set global grab for this widget.
A global grab directs all events to this and descendant widgets on the display. Use with caution - other
applications do not get events anymore.

grab_status()
Return None, “local” or “global” if this widget has no, a local or a global grab.

grid(baseWidth=None, baseHeight=None, widthInc=None, heightInc=None)
Instruct the window manager that this widget shall only be resized on grid boundaries. WIDTHINC and
HEIGHTINC are the width and height of a grid unit in pixels. BASEWIDTH and BASEHEIGHT are the
number of grid units requested in Tk_GeometryRequest.

grid_bbox(column=None, row=None, col2=None, row2=None)
Return a tuple of integer coordinates for the bounding box of this widget controlled by the geometry
manager grid.

If COLUMN, ROW is given the bounding box applies from the cell with row and column 0 to the specified
cell. If COL2 and ROW2 are given the bounding box starts at that cell.

The returned integers specify the offset of the upper left corner in the master widget and the width and
height.

grid_columnconfigure(index, cnf={}, **kw)
Configure column INDEX of a grid.
Valid resources are minsize (minimum size of the column), weight (how much does additional space
propagate to this column) and pad (how much space to let additionally).

grid_location(x, y)
Return a tuple of column and row which identify the cell at which the pixel at position X and Y inside the
master widget is located.

grid_propagate(flag=["_noarg_"])
Set or get the status for propagation of geometry information.
A boolean argument specifies whether the geometry information of the slaves will determine the size of
this widget. If no argument is given, the current setting will be returned.

grid_rowconfigure(index, cnf={}, **kw)
Configure row INDEX of a grid.
Valid resources are minsize (minimum size of the row), weight (how much does additional space propagate
to this row) and pad (how much space to let additionally).

grid_size()
Return a tuple of the number of column and rows in the grid.

grid_slaves(row=None, column=None)
Return a list of all slaves of this widget in its packing order.

group(pathName=None)
Set the group leader widgets for related widgets to PATHNAME. Return the group leader of this widget if
None is given.
iconbitmap (bitmap=None, default=None)
    Set bitmap for the iconified widget to BITMAP. Return the bitmap if None is given.
    
    Under Windows, the DEFAULT parameter can be used to set the icon for the widget and any descend-
    ents that don’t have an icon set explicitly. DEFAULT can be the relative path to a .ico file (example: 
    root.iconbitmap(default='myicon.ico') ). See Tk documentation for more information.

iconify()
    Display widget as icon.

iconmask (bitmap=None)
    Set mask for the icon bitmap of this widget. Return the mask if None is given.

iconname (newName=None)
    Set the name of the icon for this widget. Return the name if None is given.

iconposition (x=None, y=None)
    Set the position of the icon of this widget to X and Y. Return a tuple of the current values of X and X if
    None is given.

iconwindow (pathName=None)
    Set widget PATHNAME to be displayed instead of icon. Return the current value if None is given.

image_names()
    Return a list of all existing image names.

image_types()
    Return a list of all available image types (e.g. photo bitmap).

keys()
    Return a list of all resource names of this widget.

lift (aboveThis=None)
    Raise this widget in the stacking order.

lower (belowThis=None)
    Lower this widget in the stacking order.

mainloop (n=0)
    Call the mainloop of Tk.

maxsize (width=None, height=None)
    Set max WIDTH and HEIGHT for this widget. If the window is gridded the values are given in grid units.
    Return the current values if None is given.

minsize (width=None, height=None)
    Set min WIDTH and HEIGHT for this widget. If the window is gridded the values are given in grid units.
    Return the current values if None is given.

nametowidget (name)
    Return the Tkinter instance of a widget identified by its Tcl name NAME.

option_add (pattern, value, priority=None)
    Set a VALUE (second parameter) for an option PATTERN (first parameter).
    An optional third parameter gives the numeric priority (defaults to 80).

option_clear()
    Clear the option database.
    It will be reloaded if option_add is called.

option_get (name, className)
    Return the value for an option NAME for this widget with CLASSNAME.
Values with higher priority override lower values.

**option_readfile**(fileName, priority=None)
Read file FILENAME into the option database.
An optional second parameter gives the numeric priority.

**overrideredirect**(boolean=None)
Instruct the window manager to ignore this widget if BOOLEAN is given with 1. Return the current value if None is given.

**pack_propagate**(flag=["_noarg_"])
Set or get the status for propagation of geometry information.
A boolean argument specifies whether the geometry information of the slaves will determine the size of this widget. If no argument is given the current setting will be returned.

**pack_slaves**()
Return a list of all slaves of this widget in its packing order.

**place_slaves**()
Return a list of all slaves of this widget in its packing order.

**positionfrom**(who=None)
Instruct the window manager that the position of this widget shall be defined by the user if WHO is “user”, and by its own policy if WHO is “program”.

**propagate**(flag=["_noarg_"])
Set or get the status for propagation of geometry information.
A boolean argument specifies whether the geometry information of the slaves will determine the size of this widget. If no argument is given the current setting will be returned.

**protocol**(name=None, func=None)
Bind function FUNC to command NAME for this widget. Return the function bound to NAME if None is given. NAME could be e.g. “WM_SAVE_YOURSELF” or “WM_DELETE_WINDOW”.

**quit**()
Quit the Tcl interpreter. All widgets will be destroyed.

**register**(func, subst=None, needcleanup=1)
Return a newly created Tcl function. If this function is called, the Python function FUNC will be executed. An optional function SUBST can be given which will be executed before FUNC.

**resizable**(width=None, height=None)
Instruct the window manager whether this width can be resized in WIDTH or HEIGHT. Both values are boolean values.

**rowconfigure**(index, cnf={}, **kw)
Configure row INDEX of a grid.
Valid resources are minsize (minimum size of the row), weight (how much does additional space propagate to this row) and pad (how much space to let additionally).

**selection_clear**(**kw)
Clear the current X selection.

**selection_get**(**kw)
Return the contents of the current X selection.
A keyword parameter selection specifies the name of the selection and defaults to PRIMARY. A keyword parameter displayof specifies a widget on the display to use. A keyword parameter type specifies the
form of data to be fetched, defaulting to STRING except on X11, where UTF8_STRING is tried before STRING.

`selection_handle(command, **kw)`

Specify a function `COMMAND` to call if the X selection owned by this widget is queried by another application.

This function must return the contents of the selection. The function will be called with the arguments `OFFSET` and `LENGTH` which allows the chunking of very long selections. The following keyword parameters can be provided: `selection` - name of the selection (default PRIMARY), `type` - type of the selection (e.g. STRING, FILE_NAME).

`selection_own(**kw)`

Become owner of X selection.

A keyword parameter selection specifies the name of the selection (default PRIMARY).

`selection_own_get(**kw)`

Return owner of X selection.

The following keyword parameter can be provided: `selection` - name of the selection (default PRIMARY), `type` - type of the selection (e.g. STRING, FILE_NAME).

`send(interp, cmd, *args)`

Send Tcl command `CMD` to different interpreter `INTERP` to be executed.

`setvar(name='PY_VAR', value='1')`

Set Tcl variable `NAME` to `VALUE`.

`show()`

Return a tuple of the number of column and rows in the grid.

`size()`

Return a tuple of the number of column and rows in the grid.

`sizefrom(who=None)`

Instruct the window manager that the size of this widget shall be defined by the user if WHO is “user”, and by its own policy if WHO is “program”.

`slaves()`

Return a list of all slaves of this widget in its packing order.

`state(newstate=None)`

Query or set the state of this widget as one of normal, icon, iconic (see `wm_iconwindow`), withdrawn, or zoomed (Windows only).

`title(string=None)`

Set the title of this widget.

`tk_bisque()`

Change the color scheme to light brown as used in Tk 3.6 and before.

`tk_focusFollowsMouse()`

The widget under mouse will get automatically focus. Can not be disabled easily.

`tk_focusNext()`

Return the next widget in the focus order which follows widget which has currently the focus.

The focus order first goes to the next child, then to the children of the child recursively and then to the next sibling which is higher in the stacking order. A widget is omitted if it has the takefocus resource set to 0.

`tk_focusPrev()`

Return previous widget in the focus order. See `tk_focusNext` for details.
**tk_menuBar** (*args*)
Do not use. Needed in Tk 3.6 and earlier.

**tk_setPalette** (*args, **kw*)
Set a new color scheme for all widget elements.
A single color as argument will cause that all colors of Tk widget elements are derived from this. Alternatively several keyword parameters and its associated colors can be given. The following keywords are valid: activeBackground, foreground, selectColor, activeForeground, highlightBackground, selectBackground, background, highlightColor, selectForeground, disabledForeground, insertBackground, troughColor.

**tk_strictMotif** (boolean=None)
Set Tcl internal variable, whether the look and feel should adhere to Motif.
A parameter of 1 means adhere to Motif (e.g. no color change if mouse passes over slider). Returns the set value.

**tkraise** (aboveThis=None)
Raise this widget in the stacking order.

**transient** (master=None)
Instruct the window manager that this widget is transient with regard to widget MASTER.

**unbind** (sequence, funcid=None)
Unbind for this widget for event SEQUENCE the function identified with FUNCID.

**unbind_all** (sequence)
Unbind for all widgets for event SEQUENCE all functions.

**unbind_class** (className, sequence)
Unbind for all widgets with bindtag CLASSNAME for event SEQUENCE all functions.

**update**()
Enter event loop until all pending events have been processed by Tcl.

**update_idletasks**()
Enter event loop until all idle callbacks have been called. This will update the display of windows but not process events caused by the user.

**wait_variable** (name='PY_VAR')
Wait until the variable is modified.
A parameter of type IntVar, StringVar, DoubleVar or BooleanVar must be given.

**wait_visibility** (window=None)
Wait until the visibility of a WIDGET changes (e.g. it appears).
If no parameter is given self is used.

**wait_window** (window=None)
Wait until a WIDGET is destroyed.
If no parameter is given self is used.

**waitvar** (name='PY_VAR')
Wait until the variable is modified.
A parameter of type IntVar, StringVar, DoubleVar or BooleanVar must be given.

**winfo_atom** (name, displayof=0)
Return integer which represents atom NAME.
winfo_atomname(id, displayof=0)
   Return name of atom with identifier ID.

winfo_cells()
   Return number of cells in the colormap for this widget.

winfo_children()
   Return a list of all widgets which are children of this widget.

winfo_class()
   Return window class name of this widget.

winfo_colormapfull()
   Return true if at the last color request the colormap was full.

winfo_containing(rootX, rootY, displayof=0)
   Return the widget which is at the root coordinates ROOTX, ROOTY.

winfo_depth()
   Return the number of bits per pixel.

winfo_exists()
   Return true if this widget exists.

winfo_fpixels(number)
   Return the number of pixels for the given distance NUMBER (e.g. “3c”) as float.

winfo_geometry()
   Return geometry string for this widget in the form “widthxheight+X+Y”.

winfo_height()
   Return height of this widget.

winfo_id()
   Return identifier ID for this widget.

winfo_interps(displayof=0)
   Return the name of all Tcl interpreters for this display.

winfo_ismapped()
   Return true if this widget is mapped.

winfo_manager()
   Return the window manager name for this widget.

winfo_name()
   Return the name of this widget.

winfo_parent()
   Return the name of the parent of this widget.

winfo_pathname(id, displayof=0)
   Return the pathname of the widget given by ID.

winfo_pixels(number)
   Rounded integer value of winfo_fpixels.

winfo_pointerx()
   Return the x coordinate of the pointer on the root window.

winfo_pointerxy()
   Return a tuple of x and y coordinates of the pointer on the root window.
winfo_pointery()
    Return the y coordinate of the pointer on the root window.

winfo_reqheight()
    Return requested height of this widget.

winfo_reqwidth()
    Return requested width of this widget.

winfo_rgb(color)
    Return tuple of decimal values for red, green, blue for COLOR in this widget.

winfo_rootx()
    Return x coordinate of upper left corner of this widget on the root window.

winfo_rooty()
    Return y coordinate of upper left corner of this widget on the root window.

winfo_screen()
    Return the screen name of this widget.

winfo_screencells()
    Return the number of the cells in the colormap of the screen of this widget.

winfo_screendepth()
    Return the number of bits per pixel of the root window of the screen of this widget.

winfo_screenheight()
    Return the number of pixels of the height of the screen of this widget in pixel.

winfo_screenmmheight()
    Return the number of pixels of the height of the screen of this widget in mm.

winfo_screenmmwidth()
    Return the number of pixels of the width of the screen of this widget in mm.

winfo_screencolor()
    Return one of the strings directcolor, grayscale, pseudocolor, staticcolor, staticgray, or truecolor for the
    default colormodel of this screen.

winfo_screendepth()
    Return the number of pixels of the width of the screen of this widget.

winfo_server()
    Return information of the X-Server of the screen of this widget in the form “XmajorRminor vendor vendorVersion”.

winfo_toplevel()
    Return the toplevel widget of this widget.

winfo_viewable()
    Return true if the widget and all its higher ancestors are mapped.

winfo_visual()
    Return one of the strings directcolor, grayscale, pseudocolor, staticcolor, staticgray, or truecolor for the
    colormodel of this widget.

winfo_visualid()
    Return the X identifier for the visual for this widget.

winfo_visualsavailable(includeids=0)
    Return a list of all visuals available for the screen of this widget.
Each item in the list consists of a visual name (see winfo_visual), a depth and if INCLUDEIDS=1 is given also the X identifier.

`winfo_vrootheight()`
Return the height of the virtual root window associated with this widget in pixels. If there is no virtual root window return the height of the screen.

`winfo_vrootwidth()`
Return the width of the virtual root window associated with this widget in pixel. If there is no virtual root window return the width of the screen.

`winfo_vrootx()`
Return the x offset of the virtual root relative to the root window of the screen of this widget.

`winfo_vrooty()`
Return the y offset of the virtual root relative to the root window of the screen of this widget.

`winfo_width()`
Return the width of this widget.

`winfo_x()`
Return the x coordinate of the upper left corner of this widget in the parent.

`winfo_y()`
Return the y coordinate of the upper left corner of this widget in the parent.

`withdraw()`
Withdraw this widget from the screen such that it is unmapped and forgotten by the window manager. Re-draw it with wm_deiconify.

`wm_aspect(minNumer=None, minDenom=None, maxNumer=None, maxDenom=None)`
Instruct the window manager to set the aspect ratio (width/height) of this widget to be between MINNUMER/MINDENOM and MAXNUMER/MAXDENOM. Return a tuple of the actual values if no argument is given.

`wm_attributes(*args)`
This subcommand returns or sets platform specific attributes

The first form returns a list of the platform specific flags and their values. The second form returns the value for the specific option. The third form sets one or more of the values. The values are as follows:

On Windows, -disabled gets or sets whether the window is in a disabled state. -toolwindow gets or sets the style of the window to toolwindow (as defined in the MSDN). -topmost gets or sets whether this is a topmost window (displays above all other windows).

On Macintosh, XXXXX

On Unix, there are currently no special attribute values.

`wm_client(name=None)`
Store NAME in WM_CLIENT_MACHINE property of this widget. Return current value.

`wm_colormapwindows(*wlist)`
Store list of window names (WLIST) into WM_COLORMAPWINDOWS property of this widget. This list contains windows whose colormaps differ from their parents. Return current list of widgets if WLIST is empty.

`wm_command(value=None)`
Store VALUE in WM_COMMAND property. It is the command which shall be used to invoke the application. Return current command if VALUE is None.
wm_deiconify()
    Deiconify this widget. If it was never mapped it will not be mapped. On Windows it will raise this widget
    and give it the focus.

wm_focusmodel (model=None)
    Set focus model to MODEL. “active” means that this widget will claim the focus itself, “passive” means
    that the window manager shall give the focus. Return current focus model if MODEL is None.

wm_frame()
    Return identifier for decorative frame of this widget if present.

wm_geometry (newGeometry=None)
    Set geometry to NEWGEOMETRY of the form =widthxheight+x+y. Return current value if None is given.

wm_grid (baseWidth=None, baseHeight=None, widthInc=None, heightInc=None)
    Instruct the window manager that this widget shall only be resized on grid boundaries. WIDTHINC and
    HEIGHTINC are the width and height of a grid unit in pixels. BASEWIDTH and BASEHEIGHT are the
    number of grid units requested in Tk_GeometryRequest.

wm_group (pathName=None)
    Set the group leader widgets for related widgets to PATHNAME. Return the group leader of this widget if
    None is given.

wm_iconbitmap (bitmap=None, default=None)
    Set bitmap for the iconified widget to BITMAP. Return the bitmap if None is given.
    Under Windows, the DEFAULT parameter can be used to set the icon for the widget and any descen-
    dents that don’t have an icon set explicitly. DEFAULT can be the relative path to a .ico file (example:
    root.iconbitmap(default='myicon.ico')). See Tk documentation for more information.

wm_iconify()
    Display widget as icon.

wm_iconmask (bitmap=None)
    Set mask for the icon bitmap of this widget. Return the mask if None is given.

wm_iconname (newName=None)
    Set the name of the icon for this widget. Return the name if None is given.

wm_iconposition (x=None, y=None)
    Set the position of the icon of this widget to X and Y. Return a tuple of the current values of X and X if
    None is given.

wm_iconwindow (pathName=None)
    Set widget PATHNAME to be displayed instead of icon. Return the current value if None is given.

wm_maxsize (width=None, height=None)
    Set max WIDTH and HEIGHT for this widget. If the window is gridded the values are given in grid units.
    Return the current values if None is given.

wm_minsize (width=None, height=None)
    Set min WIDTH and HEIGHT for this widget. If the window is gridded the values are given in grid units.
    Return the current values if None is given.

wm_overrideredirect (boolean=None)
    Instruct the window manager to ignore this widget if BOOLEAN is given with 1. Return the current value
    if None is given.

wm_positionfrom (who=None)
    Instruct the window manager that the position of this widget shall be defined by the user if WHO is “user”,
    and by its own policy if WHO is “program”.

Chapter 4. All packages
wm_protocol (name=None, func=None)
    Bind function FUNC to command NAME for this widget. Return the function bound to NAME if None is given. NAME could be e.g. “WM_SAVE_YOURSELF” or “WM_DELETE_WINDOW”.

wm_resizable (width=None, height=None)
    Instruct the window manager whether this width can be resized in WIDTH or HEIGHT. Both values are boolean values.

wm_sizefrom (who=None)
    Instruct the window manager that the size of this widget shall be defined by the user if WHO is “user”, and by its own policy if WHO is “program”.

wm_state (newstate=None)
    Query or set the state of this widget as one of normal, icon, iconic (see wm_iconwindow), withdrawn, or zoomed (Windows only).

wm_title (string=None)
    Set the title of this widget.

wm_transient (master=None)
    Instruct the window manager that this widget is transient with regard to widget MASTER.

wm_withdraw ()
    Withdraw this widget from the screen such that it is unmapped and forgotten by the window manager. Re-draw it with wm_deiconify.

class robot.libraries.dialogs_py.SelectionDialog (message, values)
    Bases: robot.libraries.dialogs_py._TkDialog

    after (ms, func=None, *args)
        Call function once after given time.
        MS specifies the time in milliseconds. FUNC gives the function which shall be called. Additional parameters are given as parameters to the function call. Return identifier to cancel scheduling with after_cancel.

    after_cancel (id)
        Cancel scheduling of function identified with ID.
        Identifier returned by after or after_idle must be given as first parameter.

    after_idle (func, *args)
        Call FUNC once if the Tcl main loop has no event to process.
        Return an identifier to cancel the scheduling with after_cancel.

    aspect (minNumer=None, minDenom=None, maxNumer=None, maxDenom=None)
        Instruct the window manager to set the aspect ratio (width/height) of this widget to be between MINNUMER/MINDENOM and MAXNUMER/MAXDENOM. Return a tuple of the actual values if no argument is given.

    attributes (*args)
        This subcommand returns or sets platform specific attributes
        The first form returns a list of the platform specific flags and their values. The second form returns the value for the specific option. The third form sets one or more of the values. The values are as follows:
        On Windows, -disabled gets or sets whether the window is in a disabled state. -toolwindow gets or sets the style of the window to toolwindow (as defined in the MSDN). -topmost gets or sets whether this is a topmost window (displays above all other windows).
        On Macintosh, XXXXX
        On Unix, there are currently no special attribute values.
bbox \( (\text{column}=\text{None}, \text{row}=\text{None}, \text{col2}=\text{None}, \text{row2}=\text{None}) \)

Return a tuple of integer coordinates for the bounding box of this widget controlled by the geometry manager grid.

If COLUMN, ROW is given the bounding box applies from the cell with row and column 0 to the specified cell. If COL2 and ROW2 are given the bounding box starts at that cell.

The returned integers specify the offset of the upper left corner in the master widget and the width and height.

bell \( (\text{displayof}=0) \)

Ring a display’s bell.

bind \( (\text{sequence}=\text{None}, \text{func}=\text{None}, \text{add}=\text{None}) \)

Bind to this widget at event SEQUENCE a call to function FUNC.

SEQUENCE is a string of concatenated event patterns. An event pattern is of the form \(<\text{MODIFIER-MODIFIER-TYPE-DETAIL}>\) where MODIFIER is one of Control, Mod2, Shift, Mod3, M3, Lock, Mod4, M4, Button1, B1, Mod5, M5 Button2, B2, Meta, M, Button3, B3, Alt, Button4, B4, Double, Button5, B5 Triple, Mod1, M1. TYPE is one of Activate, Enter, Map, ButtonPress, Button, Expose, Motion, ButtonRelease FocusIn, MouseWheel, Circulate, FocusOut, Property, Colormap, Gravity Reparent, Configure, KeyPress, Key, Unmap, Deactivate, KeyRelease Visibility, Destroy, Leave and DETAIL is the button number for ButtonPress, ButtonRelease and DETAIL is the Keysym for KeyPress and KeyRelease. Examples are \(<\text{Control-Button-1}>\) for pressing Control and mouse button 1 or \(<\text{Alt-A}>\) for pressing A and the Alt key (KeyPress can be omitted). An event pattern can also be a virtual event of the form \(<<\text{AString}>>\) where AString can be arbitrary. This event can be generated by event_generate. If events are concatenated they must appear shortly after each other.

FUNC will be called if the event sequence occurs with an instance of Event as argument. If the return value of FUNC is “break” no further bound function is invoked.

An additional boolean parameter ADD specifies whether FUNC will be called additionally to the other bound function or whether it will replace the previous function.

Bind will return an identifier to allow deletion of the bound function with unbind without memory leak.

If FUNC or SEQUENCE is omitted the bound function or list of bound events are returned.

bind_all \( (\text{sequence}=\text{None}, \text{func}=\text{None}, \text{add}=\text{None}) \)

Bind to all widgets at an event SEQUENCE a call to function FUNC. An additional boolean parameter ADD specifies whether FUNC will be called additionally to the other bound function or whether it will replace the previous function. See bind for the return value.

bind_class \( (\text{className}, \text{sequence}=\text{None}, \text{func}=\text{None}, \text{add}=\text{None}) \)

Bind to widgets with bindtag CLASSNAME at event SEQUENCE a call of function FUNC. An additional boolean parameter ADD specifies whether FUNC will be called additionally to the other bound function or whether it will replace the previous function. See bind for the return value.

bindtags \( (\text{tagList}=\text{None}) \)

Set or get the list of bindtags for this widget.

With no argument return the list of all bindtags associated with this widget. With a list of strings as argument the bindtags are set to this list. The bindtags determine in which order events are processed (see bind).

cget \( (\text{key}) \)

Return the resource value for a KEY given as string.

client \( (\text{name}=\text{None}) \)

Store NAME in WM_CLIENT_MACHINE property of this widget. Return current value.
**clipboard_append**(string, **kw)
Append STRING to the Tk clipboard.

A widget specified at the optional displayof keyword argument specifies the target display. The clipboard can be retrieved with selection_get.

**clipboard_clear**(**kw)
Clear the data in the Tk clipboard.

A widget specified for the optional displayof keyword argument specifies the target display.

**clipboard_get**(**kw)
Retrieve data from the clipboard on window’s display.

The window keyword defaults to the root window of the Tkinter application.

The type keyword specifies the form in which the data is to be returned and should be an atom name such as STRING or FILE_NAME. Type defaults to STRING, except on X11, where the default is to try UTF8_STRING and fall back to STRING.

This command is equivalent to:

```
selection_get(CLIPBOARD)
```

**colormapwindows**( *wlist*)
Store list of window names (WLIST) into WM_COLORMAPWINDOWS property of this widget. This list contains windows whose colormaps differ from their parents. Return current list of widgets if WLIST is empty.

**colormodel**(value=None)
Useless. Not implemented in Tk.

**columnconfigure**(index, cnf={}, **kw)
Configure column INDEX of a grid.

Valid resources are minsize (minimum size of the column), weight (how much does additional space propagate to this column) and pad (how much space to let additionally).

**command**(value=None)
Store VALUE in WM_COMMAND property. It is the command which shall be used to invoke the application. Return current command if VALUE is None.

**config**(cnf=None, **kw)
Configure resources of a widget.

The values for resources are specified as keyword arguments. To get an overview about the allowed keyword arguments call the method keys.

**configure**(cnf=None, **kw)
Configure resources of a widget.

The values for resources are specified as keyword arguments. To get an overview about the allowed keyword arguments call the method keys.

**deiconify**()
Deiconify this widget. If it was never mapped it will not be mapped. On Windows it will raise this widget and give it the focus.

**deletecommand**(name)
Internal function.

Delete the Tcl command provided in NAME.
**Robot Framework Documentation, Release 2.7.7**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>destroy</strong>()</td>
<td>Destroy this and all descendants widgets.</td>
</tr>
<tr>
<td><strong>event_add</strong>(*virtual, <em>sequences</em>)</td>
<td>Bind a virtual event <em>VIRTUAL</em> (of the form &lt;&lt;Name&gt;&gt;) to an event <em>SEQUENCE</em> such that the virtual event is triggered whenever <em>SEQUENCE</em> occurs.</td>
</tr>
<tr>
<td><strong>event_delete</strong>(*virtual, <em>sequences</em>)</td>
<td>Unbind a virtual event <em>VIRTUAL</em> from <em>SEQUENCE</em>.</td>
</tr>
<tr>
<td><strong>event_generate</strong>(*sequence, *<em>kw</em>)</td>
<td>Generate an event <em>SEQUENCE</em>. Additional keyword arguments specify parameter of the event (e.g. x, y, rootx, rooty).</td>
</tr>
<tr>
<td><strong>event_info</strong>(<em>virtual=None</em>)</td>
<td>Return a list of all virtual events or the information about the <em>SEQUENCE</em> bound to the virtual event <em>VIRTUAL</em>.</td>
</tr>
<tr>
<td><strong>focus</strong>()</td>
<td>Direct input focus to this widget.</td>
</tr>
<tr>
<td></td>
<td>If the application currently does not have the focus this widget will get the focus if the application gets the focus through the window manager.</td>
</tr>
<tr>
<td><strong>focus_displayof</strong>()</td>
<td>Return the widget which has currently the focus on the display where this widget is located.</td>
</tr>
<tr>
<td></td>
<td>Return None if the application does not have the focus.</td>
</tr>
<tr>
<td><strong>focus_force</strong>()</td>
<td>Direct input focus to this widget even if the application does not have the focus. Use with caution!</td>
</tr>
<tr>
<td><strong>focus_get</strong>()</td>
<td>Return the widget which has currently the focus in the application.</td>
</tr>
<tr>
<td></td>
<td>Use focus_displayof to allow working with several displays. Return None if application does not have the focus.</td>
</tr>
<tr>
<td><strong>focus_lastfor</strong>()</td>
<td>Return the widget which would have the focus if top level for this widget gets the focus from the window manager.</td>
</tr>
<tr>
<td><strong>focus_set</strong>()</td>
<td>Direct input focus to this widget.</td>
</tr>
<tr>
<td></td>
<td>If the application currently does not have the focus this widget will get the focus if the application gets the focus through the window manager.</td>
</tr>
<tr>
<td><strong>focusmodel</strong>(<em>model=None</em>)</td>
<td>Set focus model to <em>MODEL</em>. “active” means that this widget will claim the focus itself, “passive” means that the window manager shall give the focus. Return current focus model if <em>MODEL</em> is None.</td>
</tr>
<tr>
<td><strong>frame</strong>()</td>
<td>Return identifier for decorative frame of this widget if present.</td>
</tr>
<tr>
<td><strong>geometry</strong>(<em>newGeometry=None</em>)</td>
<td>Set geometry to <em>NEWGEOMETRY</em> of the form =widthxheight+x+y. Return current value if <em>None</em> is given.</td>
</tr>
<tr>
<td><strong>getboolean</strong>(<em>s</em>)</td>
<td>Return a boolean value for Tcl boolean values true and false given as parameter.</td>
</tr>
<tr>
<td><strong>getdouble</strong></td>
<td>alias of float</td>
</tr>
</tbody>
</table>
getint
   alias of int

goingvar(name='PY_VAR')
   Return value of Tcl variable NAME.

grab_current()
   Return widget which has currently the grab in this application or None.

grab_release()
   Release grab for this widget if currently set.

grab_set()
   Set grab for this widget.
   A grab directs all events to this and descendant widgets in the application.

grab_set_global()
   Set global grab for this widget.
   A global grab directs all events to this and descendant widgets on the display. Use with caution - other applications do not get events anymore.

grab_status()
   Return None, “local” or “global” if this widget has no, a local or a global grab.

grid(baseWidth=None, baseHeight=None, widthInc=None, heightInc=None)
   Instruct the window manager that this widget shall only be resized on grid boundaries. WIDTHINC and HEIGHTINC are the width and height of a grid unit in pixels. BASEWIDTH and BASEHEIGHT are the number of grid units requested in Tk_GeometryRequest.

grid_bbox(column=None, row=None, col2=None, row2=None)
   Return a tuple of integer coordinates for the bounding box of this widget controlled by the geometry manager grid.
   If COLUMN, ROW is given the bounding box applies from the cell with row and column 0 to the specified cell. If COL2 and ROW2 are given the bounding box starts at that cell.
   The returned integers specify the offset of the upper left corner in the master widget and the width and height.

grid_columnconfigure(index, cnf={}, **kw)
   Configure column INDEX of a grid.
   Valid resources are minsize (minimum size of the column), weight (how much does additional space propagate to this column) and pad (how much space to let additionally).

grid_location(x, y)
   Return a tuple of column and row which identify the cell at which the pixel at position X and Y inside the master widget is located.

grid_propagate(flag=['_noarg_'])
   Set or get the status for propagation of geometry information.
   A boolean argument specifies whether the geometry information of the slaves will determine the size of this widget. If no argument is given, the current setting will be returned.

grid_rowconfigure(index, cnf={}, **kw)
   Configure row INDEX of a grid.
   Valid resources are minsize (minimum size of the row), weight (how much does additional space propagate to this row) and pad (how much space to let additionally).
grid_size()
   Return a tuple of the number of column and rows in the grid.

grid_slaves (row=None, column=None)
   Return a list of all slaves of this widget in its packing order.

group (pathName=None)
   Set the group leader widgets for related widgets to PATHNAME. Return the group leader of this widget if
   None is given.

iconbitmap (bitmap=None, default=None)
   Set bitmap for the iconified widget to BITMAP. Return the bitmap if None is given.

   Under Windows, the DEFAULT parameter can be used to set the icon for the widget and any descen-
   dants that don’t have an icon set explicitly. DEFAULT can be the relative path to a .ico file (example:
   root.iconbitmap(default=’myicon.ico’ ) ). See Tk documentation for more information.

iconify()
   Display widget as icon.

iconmask (bitmap=None)
   Set mask for the icon bitmap of this widget. Return the mask if None is given.

iconname (newName=None)
   Set the name of the icon for this widget. Return the name if None is given.

iconposition (x=None, y=None)
   Set the position of the icon of this widget to X and Y. Return a tuple of the current values of X and X if
   None is given.

iconwindow (pathName=None)
   Set widget PATHNAME to be displayed instead of icon. Return the current value if None is given.

image_names ()
   Return a list of all existing image names.

image_types ()
   Return a list of all available image types (e.g. photo bitmap).

keys ()
   Return a list of all resource names of this widget.

lift (aboveThis=None)
   Raise this widget in the stacking order.

lower (belowThis=None)
   Lower this widget in the stacking order.

mainloop (n=0)
   Call the mainloop of Tk.

maxsize (width=None, height=None)
   Set max WIDTH and HEIGHT for this widget. If the window is gridded the values are given in grid units.
   Return the current values if None is given.

minsize (width=None, height=None)
   Set min WIDTH and HEIGHT for this widget. If the window is gridded the values are given in grid units.
   Return the current values if None is given.

nametowidget (name)
   Return the Tkinter instance of a widget identified by its Tcl name NAME.
option_add(pattern, value, priority=None)
Set a VALUE (second parameter) for an option PATTERN (first parameter).
An optional third parameter gives the numeric priority (defaults to 80).

option_clear()
Clear the option database.
It will be reloaded if option_add is called.

option_get(name, className)
Return the value for an option NAME for this widget with CLASSNAME.
Values with higher priority override lower values.

option_readfile(fileName, priority=None)
Read file FILENAME into the option database.
An optional second parameter gives the numeric priority.

overrideredirect(boolean=None)
Instruct the window manager to ignore this widget if BOOLEAN is given with 1. Return the current value if None is given.

pack_propagate(flag='[_noarg_]')
Set or get the status for propagation of geometry information.
A boolean argument specifies whether the geometry information of the slaves will determine the size of this widget. If no argument is given the current setting will be returned.

pack_slaves()
Return a list of all slaves of this widget in its packing order.

place_slaves()
Return a list of all slaves of this widget in its packing order.

positionfrom(who=None)
Instruct the window manager that the position of this widget shall be defined by the user if WHO is “user”, and by its own policy if WHO is “program”.

propagate(flag='[_noarg_]')
Set or get the status for propagation of geometry information.
A boolean argument specifies whether the geometry information of the slaves will determine the size of this widget. If no argument is given the current setting will be returned.

protocol(name=None, func=None)
Bind function FUNC to command NAME for this widget. Return the function bound to NAME if None is given. NAME could be e.g. “WM_SAVE_YOURSELF” or “WM_DELETE_WINDOW”.

quit()
Quit the Tcl interpreter. All widgets will be destroyed.

register(func, subst=None, needcleanup=1)
Return a newly created Tcl function. If this function is called, the Python function FUNC will be executed. An optional function SUBST can be given which will be executed before FUNC.

resizable(width=None, height=None)
Instruct the window manager whether this width can be resized in WIDTH or HEIGHT. Both values are boolean values.

rowconfigure(index, cnf={}, **kw)
Configure row INDEX of a grid.
Valid resources are minsize (minimum size of the row), weight (how much does additional space propagate to this row) and pad (how much space to let additionally).

**selection_clear(**k**w)
Clear the current X selection.

**selection_get(**k**w)
Return the contents of the current X selection.

A keyword parameter selection specifies the name of the selection and defaults to PRIMARY. A keyword parameter displayof specifies a widget on the display to use. A keyword parameter type specifies the form of data to be fetched, defaulting to STRING except on X11, where UTF8_STRING is tried before STRING.

**selection_handle**(command, **k**w)
Specify a function COMMAND to call if the X selection owned by this widget is queried by another application.

This function must return the contents of the selection. The function will be called with the arguments OFFSET and LENGTH which allows the chunking of very long selections. The following keyword parameters can be provided: selection - name of the selection (default PRIMARY), type - type of the selection (e.g. STRING, FILE_NAME).

**selection_own(**k**w)
Become owner of X selection.

A keyword parameter selection specifies the name of the selection (default PRIMARY).

**selection_own_get(**k**w)
Return owner of X selection.

The following keyword parameter can be provided: selection - name of the selection (default PRIMARY), type - type of the selection (e.g. STRING, FILE_NAME).

**send**(interp, cmd, *args)
Send Tcl command CMD to different interpreter INTERP to be executed.

**setvar**(name='PY_VAR', value='1')
Set Tcl variable NAME to VALUE.

**show**()

**size**()
Return a tuple of the number of column and rows in the grid.

**sizefrom**(who=None)
Instruct the window manager that the size of this widget shall be defined by the user if WHO is “user”, and by its own policy if WHO is “program”.

**slaves**()
Return a list of all slaves of this widget in its packing order.

**state**(newstate=None)
Query or set the state of this widget as one of normal, icon, iconic (see wm_iconwindow), withdrawn, or zoomed (Windows only).

**title**(string=None)
Set the title of this widget.

**tk_bisque**()
Change the color scheme to light brown as used in Tk 3.6 and before.
**tk_focusFollowsMouse()**
The widget under mouse will get automatically focus. Can not be disabled easily.

**tk_focusNext()**
Return the next widget in the focus order which follows widget which has currently the focus.

The focus order first goes to the next child, then to the children of the child recursively and then to the next sibling which is higher in the stacking order. A widget is omitted if it has the takefocus resource set to 0.

**tk_focusPrev()**
Return previous widget in the focus order. See tk_focusNext for details.

**tk_menuBar(*args)**
Do not use. Needed in Tk 3.6 and earlier.

**tk_setPalette(*args, **kw)**
Set a new color scheme for all widget elements.

A single color as argument will cause that all colors of Tk widget elements are derived from this. Alternatively several keyword parameters and its associated colors can be given. The following keywords are valid: activeBackground, foreground, selectColor, activeForeground, highlightBackground, selectBackground, background, highlightColor, selectForeground, disabledForeground, insertBackgroundColor, troughColor.

**tk_strictMotif(boolean=None)**
Set Tcl internal variable, whether the look and feel should adhere to Motif.

A parameter of 1 means adhere to Motif (e.g. no color change if mouse passes over slider). Returns the set value.

**tkraise(aboveThis=None)**
Raise this widget in the stacking order.

**transient(master=None)**
Instruct the window manager that this widget is transient with regard to widget MASTER.

**unbind(sequence, funcid=None)**
Unbind for this widget for event SEQUENCE the function identified with FUNCID.

**unbind_all(sequence)**
Unbind for all widgets for event SEQUENCE all functions.

**unbind_class(className, sequence)**
Unbind for a all widgets with bindtag CLASSNAME for event SEQUENCE all functions.

**update()**
Enter event loop until all pending events have been processed by Tcl.

**update_idletasks()**
Enter event loop until all idle callbacks have been called. This will update the display of windows but not process events caused by the user.

**wait_variable(name='PY_VAR')**
Wait until the variable is modified.

A parameter of type IntVar, StringVar, DoubleVar or BooleanVar must be given.

**wait_visibility(window=None)**
Wait until the visibility of a WIDGET changes (e.g. it appears).

If no parameter is given self is used.

**wait_window(window=None)**
Wait until a WIDGET is destroyed.
If no parameter is given self is used.

waitvar (name='PY_VAR')
Wait until the variable is modified.
A parameter of type IntVar, StringVar, DoubleVar or BooleanVar must be given.

winfo_atom (name, displayof=0)
Return integer which represents atom NAME.

winfo_atomname (id, displayof=0)
Return name of atom with identifier ID.

winfo_cells ()
Return number of cells in the colormap for this widget.

winfo_children ()
Return a list of all widgets which are children of this widget.

winfo_class ()
Return window class name of this widget.

winfo_colormapfull ()
Return true if at the last color request the colormap was full.

winfo_containing (rootX, rootY, displayof=0)
Return the widget which is at the root coordinates ROOTX, ROOTY.

winfo_depth ()
Return the number of bits per pixel.

winfo_exists ()
Return true if this widget exists.

winfo_fpixels (number)
Return the number of pixels for the given distance NUMBER (e.g. “3c”) as float.

winfo_geometry ()
Return geometry string for this widget in the form “widthxheight+X+Y”.

winfo_height ()
Return height of this widget.

winfo_id ()
Return identifier ID for this widget.

winfo_interps (displayof=0)
Return the name of all Tcl interpreters for this display.

winfo_ismapped ()
Return true if this widget is mapped.

winfo_manager ()
Return the window mananger name for this widget.

winfo_name ()
Return the name of this widget.

winfo_parent ()
Return the name of the parent of this widget.

winfo_pathname (id, displayof=0)
Return the pathname of the widget given by ID.
winfo_pixels(number)
    Rounded integer value of winfo_fpixels.

winfo_pointerx()
    Return the x coordinate of the pointer on the root window.

winfo_pointerxy()
    Return a tuple of x and y coordinates of the pointer on the root window.

winfo_pointery()
    Return the y coordinate of the pointer on the root window.

winfo_requestheight()
    Return requested height of this widget.

winfo_requestwidth()
    Return requested width of this widget.

winfo_rgb(color)
    Return tuple of decimal values for red, green, blue for COLOR in this widget.

winfo_rootx()
    Return x coordinate of upper left corner of this widget on the root window.

winfo_rooty()
    Return y coordinate of upper left corner of this widget on the root window.

winfo_screen()
    Return the screen name of this widget.

winfo_sceencells()
    Return the number of the cells in the colormap of the screen of this widget.

winfo_screendepth()
    Return the number of bits per pixel of the root window of the screen of this widget.

winfo_screenheight()
    Return the number of pixels of the height of the screen of this widget in pixel.

winfo_screennmheight()
    Return the number of pixels of the height of the screen of this widget in mm.

winfo_screennmwidth()
    Return the number of pixels of the width of the screen of this widget in mm.

winfo_screenvisual()
    Return one of the strings directcolor, grayscale, pseudocolor, staticcolor, staticgray, or truecolor for the default colormodel of this screen.

winfo_screenwidth()
    Return the number of pixels of the width of the screen of this widget in pixel.

winfo_server()
    Return information of the X-Server of the screen of this widget in the form “XmajorRminor vendor vendorVersion”.

winfo_toplevel()
    Return the toplevel widget of this widget.

winfo_viewable()
    Return true if the widget and all its higher ancestors are mapped.
winfo_visual()
Return one of the strings directcolor, grayscale, pseudocolor, staticcolor, staticgray, or truecolor for the
colormodel of this widget.

winfo_visualid()
Return the X identifier for the visual for this widget.

winfo_visualsavailable (includeids=0)
Return a list of all visuals available for the screen of this widget.
Each item in the list consists of a visual name (see winfo_visual), a depth and if INCLUDEIDS=1 is given
also the X identifier.

winfo_vrootheight ()
Return the height of the virtual root window associated with this widget in pixels. If there is no virtual root
window return the height of the screen.

winfo_vrootwidth ()
Return the width of the virtual root window associated with this widget in pixel. If there is no virtual root
window return the width of the screen.

winfo_vrootx ()
Return the x offset of the virtual root relative to the root window of the screen of this widget.

winfo_vrooty ()
Return the y offset of the virtual root relative to the root window of the screen of this widget.

winfo_width ()
Return the width of this widget.

winfo_x ()
Return the x coordinate of the upper left corner of this widget in the parent.

winfo_y ()
Return the y coordinate of the upper left corner of this widget in the parent.

withdraw ()
Withdraw this widget from the screen such that it is unmapped and forgotten by the window manager.
Re-draw it with wm_deiconify.

wm_aspect (minNumer=None, minDenom=None, maxNumer=None, maxDenom=None)
Instruct the window manager to set the aspect ratio (width/height) of this widget to be between MINNU-
MER/MINDENOM and MAXNUMER/MAXDENOM. Return a tuple of the actual values if no argument
is given.

wm_attributes (*args)
This subcommand returns or sets platform specific attributes
The first form returns a list of the platform specific flags and their values. The second form returns the
value for the specific option. The third form sets one or more of the values. The values are as follows:
On Windows, -disabled gets or sets whether the window is in a disabled state. -toolwindow gets or sets
the style of the window to toolwindow (as defined in the MSDN). -topmost gets or sets whether this is a
topmost window (displays above all other windows).
On Macintosh, XXXXX
On Unix, there are currently no special attribute values.

wm_client (name=None)
Store NAME in WM_CLIENT_MACHINE property of this widget. Return current value.
**wm_colormapwindows** (*wlist*)

Store list of window names (WLIST) into WM_COLORMAPWINDOWS property of this widget. This list contains windows whose colormaps differ from their parents. Return current list of widgets if WLIST is empty.

**wm_command** (*value=None*)

Store VALUE in WM_COMMAND property. It is the command which shall be used to invoke the application. Return current command if VALUE is None.

**wm_deiconify**()

Deiconify this widget. If it was never mapped it will not be mapped. On Windows it will raise this widget and give it the focus.

**wm_focusmodel** (*model=None*)

Set focus model to MODEL. “active” means that this widget will claim the focus itself, “passive” means that the window manager shall give the focus. Return current focus model if MODEL is None.

**wm_frame**()

Return identifier for decorative frame of this widget if present.

**wm_geometry** (*newGeometry=None*)

Set geometry to NEWGEOMETRY of the form =widthxheight+x+y. Return current value if None is given.

**wm_grid** (*baseWidth=None, baseHeight=None, widthInc=None, heightInc=None*)

Instruct the window manager that this widget shall only be resized on grid boundaries. WIDTHINC and HEIGHTINC are the width and height of a grid unit in pixels. BASEWIDTH and BASEHEIGHT are the number of grid units requested in Tk_GeometryRequest.

**wm_group** (*pathName=None*)

Set the group leader widgets for related widgets to PATHNAME. Return the group leader of this widget if None is given.

**wm_iconbitmap** (*bitmap=None, default=None*)

Set bitmap for the iconified widget to BITMAP. Return the bitmap if None is given.

Under Windows, the DEFAULT parameter can be used to set the icon for the widget and any descendants that don’t have an icon set explicitly. DEFAULT can be the relative path to a .ico file (example: root.iconbitmap(default=’myicon.ico’)). See Tk documentation for more information.

**wm_iconify**()

Display widget as icon.

**wm_iconmask** (*bitmap=None*)

Set mask for the icon bitmap of this widget. Return the mask if None is given.

**wm_iconname** (*newName=None*)

Set the name of the icon for this widget. Return the name if None is given.

**wm_iconposition** (*x=None, y=None*)

Set the position of the icon of this widget to X and Y. Return a tuple of the current values of X and X if None is given.

**wm_iconwindow** (*pathName=None*)

Set widget PATHNAME to be displayed instead of icon. Return the current value if None is given.

**wm_maxsize** (*width=None, height=None*)

Set max WIDTH and HEIGHT for this widget. If the window is gridded the values are given in grid units. Return the current values if None is given.

**wm_minsize** (*width=None, height=None*)

Set min WIDTH and HEIGHT for this widget. If the window is gridded the values are given in grid units. Return the current values if None is given.
**wm_overrideredirect** *(boolean=None)*
Instruct the window manager to ignore this widget if BOOLEAN is given with 1. Return the current value if None is given.

**wm_positionfrom** *(who=None)*
Instruct the window manager that the position of this widget shall be defined by the user if WHO is “user”, and by its own policy if WHO is “program”.

**wm_protocol** *(name=None, func=None)*
Bind function FUNC to command NAME for this widget. Return the function bound to NAME if None is given. NAME could be e.g. “WM_SAVE_YOURSELF” or “WM_DELETE_WINDOW”.

**wmResizable** *(width=None, height=None)*
Instruct the window manager whether this width can be resized in WIDTH or HEIGHT. Both values are boolean values.

**wm_sizefrom** *(who=None)*
Instruct the window manager that the size of this widget shall be defined by the user if WHO is “user”, and by its own policy if WHO is “program”.

**wm_state** *(newstate=None)*
Query or set the state of this widget as one of normal, icon, iconic (see wm_iconwindow), withdrawn, or zoomed (Windows only).

**wm_title** *(string=None)*
Set the title of this widget.

**wm_transient** *(master=None)*
Instruct the window manager that this widget is transient with regard to widget MASTER.

**wm_withdraw** *
Withdraw this widget from the screen such that it is unmapped and forgotten by the window manager. Re-draw it with wm_deiconify.

**class** robot.libraries.dialogs_py.PassFailDialog*(message, value=None)*

**after** *(ms, func=None, *args)*
Call function once after given time.

MS specifies the time in milliseconds. FUNC gives the function which shall be called. Additional parameters are given as parameters to the function call. Return identifier to cancel scheduling with after_cancel.

**after_cancel** *(id)*
Cancel scheduling of function identified with ID.

Identifier returned by after or after_idle must be given as first parameter.

**after_idle** *(func, *args)*
Call FUNC once if the Tcl main loop has no event to process.

Return an identifier to cancel the scheduling with after_cancel.

**aspect** *(minNumer=None, minDenom=None, maxNumer=None, maxDenom=None)*
Instruct the window manager to set the aspect ratio (width/height) of this widget to be between MINNUMER/MINDENOM and MAXNUMER/MAXDENOM. Return a tuple of the actual values if no argument is given.

**attributes** *(*args)*
This subcommand returns or sets platform specific attributes

The first form returns a list of the platform specific flags and their values. The second form returns the value for the specific option. The third form sets one or more of the values. The values are as follows:
On Windows, -disabled gets or sets whether the window is in a disabled state. -toolwindow gets or sets the style of the window to toolwindow (as defined in the MSDN). -topmost gets or sets whether this is a topmost window (displays above all other windows).

On Macintosh, XXXXX

On Unix, there are currently no special attribute values.

bbox (column=None, row=None, col2=None, row2=None)
Return a tuple of integer coordinates for the bounding box of this widget controlled by the geometry manager grid.

If COLUMN, ROW is given the bounding box applies from the cell with row and column 0 to the specified cell. If COL2 and ROW2 are given the bounding box starts at that cell.

The returned integers specify the offset of the upper left corner in the master widget and the width and height.

bell (displayof=0)
Ring a display’s bell.

bind (sequence=None, func=None, add=None)
Bind to this widget at event SEQUENCE a call to function FUNC.

SEQUENCE is a string of concatenated event patterns. An event pattern is of the form <MODIFIER-MODIFIER-TYPE-DETAIL> where MODIFIER is one of Control, Mod2, M2, Shift, Mod3, M3, Lock, Mod4, M4, Button1, B1, Mod5, M5 Button2, B2, Meta, M, Button3, B3, Alt, Button4, B4, Double, Button5, B5 Triple, Mod1, M1. TYPE is one of Activate, Enter, Map, ButtonPress, Button, Expose, Motion, ButtonRelease FocusIn, MouseWheel, Circulate, FocusOut, Property, Colormap, Gravity Reparent, Configure, KeyPress, Key, Unmap, Deactivate, KeyRelease Visibility, Destroy, Leave and DETAIL is the button number for ButtonPress, ButtonRelease and DETAIL is the Keysym for KeyPress and KeyRelease. Examples are <Control-Button-1> for pressing Control and mouse button 1 or <Alt-A> for pressing A and the Alt key (KeyPress can be omitted). An event pattern can also be a virtual event of the form <<ASTRING>> where ASTRING can be arbitrary. This event can be generated by event_generate. If events are concatenated they must appear shortly after each other.

FUNC will be called if the event sequence occurs with an instance of Event as argument. If the return value of FUNC is “break” no further bound function is invoked.

An additional boolean parameter ADD specifies whether FUNC will be called additionally to the other bound function or whether it will replace the previous function.

Bind will return an identifier to allow deletion of the bound function with unbind without memory leak.

If FUNC or SEQUENCE is omitted the bound function or list of bound events are returned.

bind_all (sequence=None, func=None, add=None)
Bind to all widgets at an event SEQUENCE a call to function FUNC. An additional boolean parameter ADD specifies whether FUNC will be called additionally to the other bound function or whether it will replace the previous function. See bind for the return value.

bind_class (className, sequence=None, func=None, add=None)
Bind to widgets with bindtag CLASSNAME at event SEQUENCE a call of function FUNC. An additional boolean parameter ADD specifies whether FUNC will be called additionally to the other bound function or whether it will replace the previous function. See bind for the return value.

bindtags (tagList=None)
Set or get the list of bindtags for this widget.

With no argument return the list of all bindtags associated with this widget. With a list of strings as argument the bindtags are set to this list. The bindtags determine in which order events are processed (see bind).
cget (key)
Return the resource value for a KEY given as string.

client (name=None)
Store NAME in WM_CLIENT_MACHINE property of this widget. Return current value.

clipboard_append (string, **kw)
Append STRING to the Tk clipboard.
A widget specified at the optional displayof keyword argument specifies the target display. The clipboard can be retrieved with selection_get.

clipboard_clear (**kw)
Clear the data in the Tk clipboard.
A widget specified for the optional displayof keyword argument specifies the target display.

clipboard_get (**kw)
Retrieve data from the clipboard on window’s display.
The window keyword defaults to the root window of the Tkinter application.
The type keyword specifies the form in which the data is to be returned and should be an atom name such as STRING or FILE_NAME. Type defaults to STRING, except on X11, where the default is to try UTF8_STRING and fall back to STRING.
This command is equivalent to:
selection_get(CLIPBOARD)

colormapwindows (*wlist)
Store list of window names (WLIST) into WM_COLORMAPWINDOWS property of this widget. This list contains windows whose colormaps differ from their parents. Return current list of widgets if WLIST is empty.

colormodel (value=None)
Useless. Not implemented in Tk.

columnconfigure (index, cnf={}, **kw)
Configure column INDEX of a grid.
Valid resources are minsize (minimum size of the column), weight (how much does additional space propagate to this column) and pad (how much space to let additionally).

command (value=None)
Store VALUE in WM_COMMAND property. It is the command which shall be used to invoke the application. Return current command if VALUE is None.

config (cnf=None, **kw)
Configure resources of a widget.
The values for resources are specified as keyword arguments. To get an overview about the allowed keyword arguments call the method keys.

configure (cnf=None, **kw)
Configure resources of a widget.
The values for resources are specified as keyword arguments. To get an overview about the allowed keyword arguments call the method keys.

deiconify ()
Deiconify this widget. If it was never mapped it will not be mapped. On Windows it will raise this widget and give it the focus.
**deletecommand** *(name)*

Internal function.

Delete the Tcl command provided in NAME.

**destroy**

Destroy this and all descendants widgets.

**event_add** *(virtual, *sequences)*

Bind a virtual event VIRTUAL (of the form <<Name>>) to an event SEQUENCE such that the virtual event is triggered whenever SEQUENCE occurs.

**event_delete** *(virtual, *sequences)*

Unbind a virtual event VIRTUAL from SEQUENCE.

**event_generate** *(sequence, **kw)*

Generate an event SEQUENCE. Additional keyword arguments specify parameter of the event (e.g. x, y, rootx, rooty).

**event_info** *(virtual=None)*

Return a list of all virtual events or the information about the SEQUENCE bound to the virtual event VIRTUAL.

**focus**

Direct input focus to this widget.

If the application currently does not have the focus this widget will get the focus if the application gets the focus through the window manager.

**focus_displayof**

Return the widget which has currently the focus on the display where this widget is located.

Return None if the application does not have the focus.

**focus_force**

Direct input focus to this widget even if the application does not have the focus. Use with caution!

**focus_get**

Return the widget which has currently the focus in the application.

Use focus_displayof to allow working with several displays. Return None if application does not have the focus.

**focus_lastfor**

Return the widget which would have the focus if top level for this widget gets the focus from the window manager.

**focus_set**

Direct input focus to this widget.

If the application currently does not have the focus this widget will get the focus if the application gets the focus through the window manager.

**focusmodel** *(model=None)*

Set focus model to MODEL. “active” means that this widget will claim the focus itself, “passive” means that the window manager shall give the focus. Return current focus model if MODEL is None.

**frame**

Return identifier for decorative frame of this widget if present.

**geometry** *(newGeometry=None)*

Set geometry to NEWGEOMETRY of the form =widthxheight+x+y. Return current value if None is given.
getboolean(s)
    Return a boolean value for Tcl boolean values true and false given as parameter.

getdouble
    alias of float

getint
    alias of int

getvar(name='PY_VAR')
    Return value of Tcl variable NAME.

grab_current()
    Return widget which has currently the grab in this application or None.

grab_release()
    Release grab for this widget if currently set.

grab_set()
    Set grab for this widget.
    A grab directs all events to this and descendant widgets in the application.

grab_set_global()
    Set global grab for this widget.
    A global grab directs all events to this and descendant widgets on the display. Use with caution - other
    applications do not get events anymore.

grab_status()
    Return None, “local” or “global” if this widget has no, a local or a global grab.

grid(baseWidth=None, baseHeight=None, widthInc=None, heightInc=None)
    Instruct the window manager that this widget shall only be resized on grid boundaries. WIDTHINC and
    HEIGHTINC are the width and height of a grid unit in pixels. BASEWIDTH and BASEHEIGHT are the
    number of grid units requested in Tk_GeometryRequest.

grid_bbox(column=None, row=None, col2=None, row2=None)
    Return a tuple of integer coordinates for the bounding box of this widget controlled by the geometry
    manager grid.
    If COLUMN, ROW is given the bounding box applies from the cell with row and column 0 to the specified
    cell. If COL2 and ROW2 are given the bounding box starts at that cell.
    The returned integers specify the offset of the upper left corner in the master widget and the width and
    height.

grid_columnconfigure(index, cnf={}, **kw)
    Configure column INDEX of a grid.
    Valid resources are minsize (minimum size of the column), weight (how much does additional space
    propagate to this column) and pad (how much space to let additionally).

grid_location(x, y)
    Return a tuple of column and row which identify the cell at which the pixel at position X and Y inside the
    master widget is located.

grid_propagate(flag=['_noarg_'])
    Set or get the status for propagation of geometry information.
    A boolean argument specifies whether the geometry information of the slaves will determine the size of
    this widget. If no argument is given, the current setting will be returned.
grid_rowconfigure(index, cnf={}, **kw)
Configure row INDEX of a grid.
Valid resources are minsize (minimum size of the row), weight (how much does additional space propagate to this row) and pad (how much space to let additionally).

grid_size()
Return a tuple of the number of column and rows in the grid.

grid_slaves(row=None, column=None)
Return a list of all slaves of this widget in its packing order.

group(pathName=None)
Set the group leader widgets for related widgets to PATHNAME. Return the group leader of this widget if None is given.

iconbitmap(bitmap=None, default=None)
Set bitmap for the iconified widget to BITMAP. Return the bitmap if None is given.
Under Windows, the DEFAULT parameter can be used to set the icon for the widget and any descendants that don’t have an icon set explicitly. DEFAULT can be the relative path to a .ico file (example: root.iconbitmap(default='myicon.ico')). See Tk documentation for more information.

iconify()
Display widget as icon.

iconmask(bitmap=None)
Set mask for the icon bitmap of this widget. Return the mask if None is given.

iconname(newName=None)
Set the name of the icon for this widget. Return the name if None is given.

iconposition(x=None, y=None)
Set the position of the icon of this widget to X and Y. Return a tuple of the current values of X and X if None is given.

iconwindow(pathName=None)
Set widget PATHNAME to be displayed instead of icon. Return the current value if None is given.

image_names()
Return a list of all existing image names.

image_types()
Return a list of all available image types (e.g. photo bitmap).

keys()
Return a list of all resource names of this widget.

lift(aboveThis=None)
Raise this widget in the stacking order.

lower(belowThis=None)
Lower this widget in the stacking order.

mainloop(n=0)
Call the mainloop of Tk.

maxsize(width=None, height=None)
Set max WIDTH and HEIGHT for this widget. If the window is gridded the values are given in grid units. Return the current values if None is given.
minsize (width=None, height=None)
Set min WIDTH and HEIGHT for this widget. If the window is gridded the values are given in grid units.
Return the current values if None is given.

nametowidget (name)
Return the Tkinter instance of a widget identified by its Tcl name NAME.

option_add (pattern, value, priority=None)
Set a VALUE (second parameter) for an option PATTERN (first parameter).
An optional third parameter gives the numeric priority (defaults to 80).

option_clear ()
Clear the option database.
It will be reloaded if option_add is called.

option_get (name, className)
Return the value for an option NAME for this widget with CLASSNAME.
Values with higher priority override lower values.

option_readfile (fileName, priority=None)
Read file FILENAME into the option database.
An optional second parameter gives the numeric priority.

overrideredirect (boolean=None)
Instruct the window manager to ignore this widget if BOOLEAN is given with 1. Return the current value if None is given.

pack_propagate (flag=['_noarg_'])
Set or get the status for propagation of geometry information.
A boolean argument specifies whether the geometry information of the slaves will determine the size of this widget. If no argument is given the current setting will be returned.

pack_slaves ()
Return a list of all slaves of this widget in its packing order.

place_slaves ()
Return a list of all slaves of this widget in its packing order.

positionfrom (who=None)
Instruct the window manager that the position of this widget shall be defined by the user if WHO is “user”, and by its own policy if WHO is “program”.

propagate (flag=['_noarg_'])
Set or get the status for propagation of geometry information.
A boolean argument specifies whether the geometry information of the slaves will determine the size of this widget. If no argument is given the current setting will be returned.

protocol (name=None, func=None)
Bind function FUNC to command NAME for this widget. Return the function bound to NAME if None is given. NAME could be e.g. “WM_SAVE_YOURSELF” or “WM_DELETE_WINDOW”.

quit ()
Quit the Tcl interpreter. All widgets will be destroyed.

register (func, subst=None, needcleanup=1)
Return a newly created Tcl function. If this function is called, the Python function FUNC will be executed. An optional function SUBST can be given which will be executed before FUNC.
resizable (width=None, height=None)
Instruct the window manager whether this width can be resized in WIDTH or HEIGHT. Both values are boolean values.

rowconfigure (index, cnf={}, **kw)
Configure row INDEX of a grid.
Valid resources are minsize (minimum size of the row), weight (how much does additional space propagate to this row) and pad (how much space to let additionally).

selection_clear (**kw)
Clear the current X selection.

selection_get (**kw)
Return the contents of the current X selection.
A keyword parameter selection specifies the name of the selection and defaults to PRIMARY. A keyword parameter displayof specifies a widget on the display to use. A keyword parameter type specifies the form of data to be fetched, defaulting to STRING except on X11, where UTF8_STRING is tried before STRING.

selection_handle (command, **kw)
Specify a function COMMAND to call if the X selection owned by this widget is queried by another application.
This function must return the contents of the selection. The function will be called with the arguments OFFSET and LENGTH which allows the chunking of very long selections. The following keyword parameters can be provided: selection - name of the selection (default PRIMARY), type - type of the selection (e.g. STRING, FILE_NAME).

selection_own (**kw)
Become owner of X selection.
A keyword parameter selection specifies the name of the selection (default PRIMARY).

selection_own_get (**kw)
Return owner of X selection.
The following keyword parameter can be provided: selection - name of the selection (default PRIMARY), type - type of the selection (e.g. STRING, FILE_NAME).

send (interp, cmd, *args)
Send Tcl command CMD to different interpreter INTERP to be executed.

setvar (name='PY_VAR', value='1')
Set Tcl variable NAME to VALUE.

show ()

size ()
Return a tuple of the number of column and rows in the grid.

sizefrom (who=None)
Instruct the window manager that the size of this widget shall be defined by the user if WHO is “user”, and by its own policy if WHO is “program”.

slaves ()
Return a list of all slaves of this widget in its packing order.

state (newstate=None)
Query or set the state of this widget as one of normal, icon, iconic (see wm_iconwindow), withdrawn, or zoomed (Windows only).
**title** (string=None)
Set the title of this widget.

**tk_bisque**()
Change the color scheme to light brown as used in Tk 3.6 and before.

**tk_focusFollowsMouse**()
The widget under mouse will get automatically focus. Can not be disabled easily.

**tk_focusNext**()
Return the next widget in the focus order which follows widget which has currently the focus.
The focus order first goes to the next child, then to the children of the child recursively and then to the next sibling which is higher in the stacking order. A widget is omitted if it has the takefocus resource set to 0.

**tk_focusPrev**()
Return previous widget in the focus order. See tk_focusNext for details.

**tk_menuBar**(*args)
Do not use. Needed in Tk 3.6 and earlier.

**tk_setPalette**(*args, **kw)
Set a new color scheme for all widget elements.
A single color as argument will cause that all colors of Tk widget elements are derived from this. Alternatively several keyword parameters and its associated colors can be given. The following keywords are valid: activeBackground, foreground, selectColor, activeForeground, highlightBackground, selectBackground, background, highlightColor, selectForeground, disabledForeground, insertBackground, troughColor.

**tk_strictMotif** (boolean=None)
Set Tcl internal variable, whether the look and feel should adhere to Motif.
A parameter of 1 means adhere to Motif (e.g. no color change if mouse passes over slider). Returns the set value.

**tkraise** (aboveThis=None)
Raise this widget in the stacking order.

**transient** (master=None)
Instruct the window manager that this widget is transient with regard to widget MASTER.

**unbind** (sequence, funcid=None)
Unbind for this widget for event SEQUENCE the function identified with FUNCID.

**unbind_all** (sequence)
Unbind for all widgets for event SEQUENCE all functions.

**unbind_class** (className, sequence)
Unbind for a all widgets with bindtag CLASSNAME for event SEQUENCE all functions.

**update**()
Enter event loop until all pending events have been processed by Tcl.

**update_idletasks**()
Enter event loop until all idle callbacks have been called. This will update the display of windows but not process events caused by the user.

**wait_variable** (name=’PY_VAR’)
Wait until the variable is modified.
A parameter of type IntVar, StringVar, DoubleVar or BooleanVar must be given.
wait_visibleity (window=None)
    Wait until the visibility of a WIDGET changes (e.g. it appears).
    If no parameter is given self is used.

wait_window (window=None)
    Wait until a WIDGET is destroyed.
    If no parameter is given self is used.

waitvar (name='PY_VAR')
    Wait until the variable is modified.
    A parameter of type IntVar, StringVar, DoubleVar or BooleanVar must be given.

winfo_atom (name, displayof=0)
    Return integer which represents atom NAME.

winfo_atomname (id, displayof=0)
    Return name of atom with identifier ID.

winfo_cells ()
    Return number of cells in the colormap for this widget.

winfo_children ()
    Return a list of all widgets which are children of this widget.

winfo_class ()
    Return window class name of this widget.

winfo_colormapfull ()
    Return true if at the last color request the colormap was full.

winfo_containing (rootX, rootY, displayof=0)
    Return the widget which is at the root coordinates ROOTX, ROOTY.

winfo_depth ()
    Return the number of bits per pixel.

winfo_exists ()
    Return true if this widget exists.

winfo_fpixels (number)
    Return the number of pixels for the given distance NUMBER (e.g. “3c”) as float.

winfo_geometry ()
    Return geometry string for this widget in the form “widthxheight+X+Y”.

winfo_height ()
    Return height of this widget.

winfo_id ()
    Return identifier ID for this widget.

winfo_interps (displayof=0)
    Return the name of all Tcl interpreters for this display.

winfo_ismapped ()
    Return true if this widget is mapped.

winfo_manager ()
    Return the window mananger name for this widget.

winfo_name ()
    Return the name of this widget.
winfo_parent()
    Return the name of the parent of this widget.

winfo_pathname(id, displayof=0)
    Return the pathname of the widget given by ID.

winfo_pixels(number)
    Rounded integer value of winfo_fpixels.

winfo_pointerx()
    Return the x coordinate of the pointer on the root window.

winfo_pointerxy()
    Return a tuple of x and y coordinates of the pointer on the root window.

winfo_pointery()
    Return the y coordinate of the pointer on the root window.

winfo_reqheight()
    Return requested height of this widget.

winfo_reqwidth()
    Return requested width of this widget.

winfo_rgb(color)
    Return tuple of decimal values for red, green, blue for COLOR in this widget.

winfo_rootx()
    Return x coordinate of upper left corner of this widget on the root window.

winfo_rooty()
    Return y coordinate of upper left corner of this widget on the root window.

winfo_screen()
    Return the screen name of this widget.

winfo_screencells()
    Return the number of the cells in the colormap of the screen of this widget.

winfo_screendepth()
    Return the number of bits per pixel of the root window of the screen of this widget.

winfo_screenheight()
    Return the number of pixels of the height of the screen of this widget in pixel.

winfo_screenmmheight()
    Return the number of pixels of the height of the screen of this widget in mm.

winfo_screenmmwidth()
    Return the number of pixels of the width of the screen of this widget in mm.

winfo_screenvisual()
    Return one of the strings directcolor, grayscale, pseudocolor, staticcolor, staticgray, or truecolor for the default colormodel of this screen.

winfo_screenwidth()
    Return the number of pixels of the width of the screen of this widget in pixel.

winfo_server()
    Return information of the X-Server of the screen of this widget in the form “XmajorRminor vendor vendorVersion”.

winfo_toplevel()
    Return the toplevel widget of this widget.
**winfo_viewable()**
Return true if the widget and all its higher ancestors are mapped.

**winfo_visual()**
Return one of the strings directcolor, grayscale, pseudocolor, staticcolor, staticgray, or truecolor for the colormap of this widget.

**winfo_visualid()**
Return the X identifier for the visual for this widget.

**winfo_visualsavailable (includeids=0)**
Return a list of all visuals available for the screen of this widget.

Each item in the list consists of a visual name (see winfo_visual), a depth and if INCLUDEIDS=1 is given also the X identifier.

**winfo_vrootheight ()**
Return the height of the virtual root window associated with this widget in pixels. If there is no virtual root window return the height of the screen.

**winfo_vrootwidth ()**
Return the width of the virtual root window associated with this widget in pixels. If there is no virtual root window return the width of the screen.

**winfo_vrootx ()**
Return the x offset of the virtual root relative to the root window of the screen of this widget.

**winfo_vrooty ()**
Return the y offset of the virtual root relative to the root window of the screen of this widget.

**winfo_width ()**
Return the width of this widget.

**winfo_x ()**
Return the x coordinate of the upper left corner of this widget in the parent.

**winfo_y ()**
Return the y coordinate of the upper left corner of this widget in the parent.

**withdraw ()**
Withdraw this widget from the screen such that it is unmapped and forgotten by the window manager. Re-draw it with wm_deiconify.

**wm_aspect (minNumer=None, minDenom=None, maxNumer=None, maxDenom=None)**
Instruct the window manager to set the aspect ratio (width/height) of this widget to be between MINNUMER/MINDENOM and MAXNUMER/MAXDENOM. Return a tuple of the actual values if no argument is given.

**wm_attributes (*args)**
This subcommand returns or sets platform specific attributes

The first form returns a list of the platform specific flags and their values. The second form returns the value for the specific option. The third form sets one or more of the values. The values are as follows:

On Windows, -disabled gets or sets whether the window is in a disabled state. -toolwindow gets or sets the style of the window to toolwindow (as defined in the MSDN). -topmost gets or sets whether this is a topmost window (displays above all other windows).

On Macintosh, XXXXX

On Unix, there are currently no special attribute values.
**wm_client** *(name=None)*

Store NAME in WM_CLIENT_MACHINE property of this widget. Return current value.

**wm_colormapwindows** *(wlist)*

Store list of window names (WLIST) into WM_COLORMAPWINDOWS property of this widget. This list contains windows whose colormaps differ from their parents. Return current list of widgets if WLIST is empty.

**wm_command** *(value=None)*

Store VALUE in WM_COMMAND property. It is the command which shall be used to invoke the application. Return current command if VALUE is None.

**wm_deiconify** *

Deiconify this widget. If it was never mapped it will not be mapped. On Windows it will raise this widget and give it the focus.

**wm_focusmodel** *(model=None)*

Set focus model to MODEL. “active” means that this widget will claim the focus itself, “passive” means that the window manager shall give the focus. Return current focus model if MODEL is None.

**wm_frame** *

Return identifier for decorative frame of this widget if present.

**wm_geometry** *(newGeometry=None)*

Set geometry to NEWGEOMETRY of the form =widthxheight+x+y. Return current value if None is given.

**wm_grid** *(baseWidth=None, baseHeight=None, widthInc=None, heightInc=None)*

Instruct the window manager that this widget shall only be resized on grid boundaries. WIDTHINC and HEIGHTINC are the width and height of a grid unit in pixels. BASEWIDTH and BASEHEIGHT are the number of grid units requested in Tk_GeometryRequest.

**wm_group** *(pathName=None)*

Set the group leader widgets for related widgets to PATHNAME. Return the group leader of this widget if None is given.

**wm_iconbitmap** *(bitmap=None, default=None)*

Set bitmap for the iconified widget to BITMAP. Return the bitmap if None is given.

Under Windows, the DEFAULT parameter can be used to set the icon for the widget and any descendants that don’t have an icon set explicitly. DEFAULT can be the relative path to a .ico file (example: root.iconbitmap(default='myicon.ico')). See Tk documentation for more information.

**wm_iconify** *

Display widget as icon.

**wm_iconmask** *(bitmap=None)*

Set mask for the icon bitmap of this widget. Return the mask if None is given.

**wm_iconname** *(newName=None)*

Set the name of the icon for this widget. Return the name if None is given.

**wm_iconposition** *(x=None, y=None)*

Set the position of the icon of this widget to X and Y. Return a tuple of the current values of X and X if None is given.

**wm_iconwindow** *(pathName=None)*

Set widget PATHNAME to be displayed instead of icon. Return the current value if None is given.

**wm_maxsize** *(width=None, height=None)*

Set max WIDTH and HEIGHT for this widget. If the window is gridded the values are given in grid units. Return the current values if None is given.
**wm_minsize** *(width=None, height=None)*
Set min WIDTH and HEIGHT for this widget. If the window is gridded the values are given in grid units. Return the current values if None is given.

**wm_overrideredirect** *(boolean=None)*
Instruct the window manager to ignore this widget if BOOLEAN is given with 1. Return the current value if None is given.

**wm_positionfrom** *(who=None)*
Instruct the window manager that the position of this widget shall be defined by the user if WHO is “user”, and by its own policy if WHO is “program”.

**wm_protocol** *(name=None, func=None)*
Bind function FUNC to command NAME for this widget. Return the function bound to NAME if None is given. NAME could be e.g. “WM_SAVE_YOURSELF” or “WM_DELETE_WINDOW”.

**wm_resizable** *(width=None, height=None)*
Instruct the window manager whether this width can be resized in WIDTH or HEIGHT. Both values are boolean values.

**wm_sizefrom** *(who=None)*
Instruct the window manager that the size of this widget shall be defined by the user if WHO is “user”, and by its own policy if WHO is “program”.

**wm_state** *(newstate=None)*
Query or set the state of this widget as one of normal, icon, iconic (see wm_iconwindow), withdrawn, or zoomed (Windows only).

**wm_title** *(string=None)*
Set the title of this widget.

**wm_transient** *(master=None)*
Instruct the window manager that this widget is transient with regard to widget MASTER.

**wm_withdraw** *
Withdraw this widget from the screen such that it is unmapped and forgotten by the window manager. Re-draw it with wm_deiconify.

### model Package

**model Package**

Contains base classes and other generic functionality.

In RF 2.7 this package is mainly used by robot.result package, but there is a plan to change also robot.running to use this in RF 2.8.

This package is considered stable.

### criticality Module

**class** robot.model.criticality.Criticality *(critical_tags=None, non_critical_tags=None)*
Bases: object

- **tag_is_critical**(tag)
- **tag_is_non_critical**(tag)
- **test_is_critical**(test)
filter Module

class robot.model.filter.Filter(include_suites=None, include_tests=None, include_tags=None, exclude_tags=None):
    Bases: robot.model.visitor.SuiteVisitor

    include_suites
    include_tests
    include_tags
    exclude_tags
    start_suite(suite)
    end_keyword(keyword)
    end_message(msg)
    end_suite(suite)
    end_test(test)
    start_keyword(keyword)
    start_message(msg)
    start_test(test)
    visit_message(msg)
    visit_suite(suite)
    visit_test(test)
    visit_keyword(keyword)

itemlist Module

class robot.model.itemlist.ItemList(item_class, common_attrs=None, items=None):
    Bases: object
    create(*args, **kwargs)
    append(item)
    extend(items)
    index(item)
    clear()
    visit(visitor)

keyword Module

class robot.model.keyword.Keyword(name='', doc='', args=None, type='kw', timeout=''):
    Bases: robot.model.modelobject.ModelObject

    KEYWORD_TYPE = 'kw'
    SETUP_TYPE = 'setup'
TEARDOWN_TYPE = 'teardown'
FOR_LOOP_TYPE = 'for'
FOR_ITEM_TYPE = 'foritem'

message_class
   alias of Message

parent
name
doc
args
type	timeout
keywords
messages
id
visit(visitor)

class robot.model.keyword.Keywords(keyword_class=<class 'robot.model.keyword.Keyword'>, parent=None, keywords=None)
    Bases: robot.model.itemlist.ItemList
    setup
tear down
all
append(item)
clear()
create(*args, **kwargs)
extend(items)
index(item)
visit(visitor)
normal

message Module

class robot.model.message.Message(message='', level='INFO', html=False, timestamp=None, parent=None)
    Bases: robot.model.modelobject.ModelObject
message
level
html
timestamp
parent

4.1. robot Package
html_message

visit(visitor)

class robot.model.message.Messages(message_class=<class 'robot.model.message.Message'>, parent=None, messages=None)
    Bases: robot.model.itemlist.ItemList
    append(item)
    clear()
    create(*args, **kwargs)
    extend(items)
    index(item)
    visit(visitor)

metadata Module

class robot.model.metadata.Metadata(initial=None)
    Bases: robot.utils.normalizing.NormalizedDict
    clear()
    copy()
   classmethod fromkeys(iterable, value=None)
    get(key, default=None)
    has_key(key)
    items()
    iteritems()
    iterkeys()
    itervalues()
    keys()
    pop(key)
    popitem()
    set(key, value)
    setdefault(key, failobj=None)
    update(dict=None, **kwargs)
    values()

modelobject Module

class robot.model.modelobject.ModelObject
    Bases: object
namepatterns Module

class robot.model.namepatterns.SuiteNamePatterns(patterns=None)
    Bases: robot.model.namepatterns._NamePatterns
    match(name, longname=None)

class robot.model.namepatterns.TestNamePatterns(patterns=None)
    Bases: robot.model.namepatterns._NamePatterns
    match(name, longname=None)

statistics Module

class robot.model.statistics.Statistics(suite, suite_stat_level=-1, tag_stat_include=None,
                                        tag_stat_exclude=None, tag_stat_combine=None,
                                        tag_doc=None, tag_stat_link=None)
    Bases: object
    visit(visitor)

class robot.model.statistics.StatisticsBuilder(total_builder, suite_builder, tag_builder)
    Bases: robot.model.visitor.SuiteVisitor
    start_suite(suite)
    end_suite(suite)
    visit_keyword(kw)
    end_keyword(keyword)
    end_message(msg)
    end_test(test)
    start_keyword(keyword)
    start_message(msg)
    start_test(test)
    visit_message(msg)
    visit_suite(suite)

stats Module

class robot.model.stats.Stat(name)
    Bases: object
    get_attributes(include_label=False, include_elapsed=False, exclude_empty=False,
                  values_as_strings=False, html_escape=False)
    total
    add_test(test)
    visit(visitor)
```python
class robot.model.stats.TotalStat(name):
    Bases: robot.model.stats.Stat
    type = 'total'
    add_test(test)
    get_attributes(include_label=False, include_elapsed=False, exclude_empty=False, values_as_strings=False, html_escape=False)
    total
    visit(visitor)

class robot.model.stats.SuiteStat(suite):
    Bases: robot.model.stats.Stat
    type = 'suite'
    add_stat(other)
    add_test(test)
    get_attributes(include_label=False, include_elapsed=False, exclude_empty=False, values_as_strings=False, html_escape=False)
    total
    visit(visitor)

class robot.model.stats.TagStat(name, doc='', links=None, critical=False, non_critical=False, combined=''):
    Bases: robot.model.stats.Stat
    type = 'tag'
    info
    add_test(test)
    get_attributes(include_label=False, include_elapsed=False, exclude_empty=False, values_as_strings=False, html_escape=False)
    total
    visit(visitor)

class robot.model.stats.CombinedTagStat(pattern, name=None, doc='', links=None):
    Bases: robot.model.stats.TagStat
    match(tags)
    add_test(test)
    get_attributes(include_label=False, include_elapsed=False, exclude_empty=False, values_as_strings=False, html_escape=False)
    info
    total
    type = 'tag'
    visit(visitor)
```

158 Chapter 4. All packages
suitestatistics Module

class robot.model.suitestatistics.SuiteStatistics(suite)
    Bases: object
    visit(visitor)

class robot.model.suitestatistics.SuiteStatisticsBuilder(suite_stat_level)
    Bases: object
    current
    start_suite(suite)
    add_test(test)
    end_suite()

tags Module

class robot.model.tags.Tags(tags=None)
    Bases: object
    add(tags)
    remove(tags)
    match(tags)

class robot.model.tags.TagPatterns(patterns)
    Bases: object
    match(tags)

robot.model.tags.TagPattern(pattern)
tagsetter Module

class robot.model.tagsetter.TagSetter(add=None, remove=None)
    Bases: robot.model.visitor.SuiteVisitor
    start_suite(suite)
    visit_test(test)
    visit_keyword(keyword)
    end_keyword(keyword)
    end_message(msg)
    end_suite(suite)
    end_test(test)
    start_keyword(keyword)
    start_message(msg)
    start_test(test)
    visit_message(msg)
    visit_suite(suite)
tagstatistics Module

class robot.model.tagstatistics.TagStatistics(combined_stats)
   Bases: object
   visit(visitor)

class robot.model.tagstatistics.TagStatisticsBuilder(criticality=None, included=None, excluded=None, combined=None, docs=None, links=None)
   Bases: object
   add_test(test)

class robot.model.tagstatistics.TagStatInfo(criticality=None, docs=None, links=None)
   Bases: object
   get_stat(tag)
   get_combined_stats(combined=None)
   get_combined_stat(pattern, name=None)
   get_doc(tag)
   get_links(tag)

class robot.model.tagstatistics.TagStatDoc(pattern, doc)
   Bases: object
   match(tag)

class robot.model.tagstatistics.TagStatLink(pattern, link, title)
   Bases: object
   match(tag)
   get_link(tag)

testcase Module

class robot.model.testcase.TestCase(name='', doc='', tags=None, timeout='')
   Bases: robot.model.modelobject.ModelObject
      keyword_class
         alias of Keyword

parent
name
doc
timeout
tags
keywords
id
longname
critical
visit (visitor)

class robot.model.testcase.TestCases (test_class=<class 'robot.model.testcase.TestCase'>, parent=None, tests=None)
    Bases: robot.model.itemlist.ItemList
    append (item)
    create (*args, **kwargs)
    extend (items)
    index (item)
    visit (visitor)

testsuite Module

class robot.model.testsuite.TestSuite (source='', name='', doc='', metadata=None)
    Bases: robot.model.modelobject.ModelObject
    test_class
        alias of TestCase
    keyword_class
        alias of Keyword
    parent
    source
    doc
    name
    set_criticality (critical_tags=None, non_critical_tags=None)
    criticality
    metadata
    suites
    tests
    keywords
    id
    longname
    test_count
    set_tags (add=None, remove=None)
    filter (included_suites=None, included_tests=None, included_tags=None, excluded_tags=None)
    visit (visitor)

class robot.model.testsuite.TestSuites (suite_class=<class 'robot.model.testsuite.TestSuite'>, parent=None, suites=None)
    Bases: robot.model.itemlist.ItemList
    append (item)
clear()
create(*args, **kwargs)
extend(items)
index(item)
visit(visitor)

totalstatistics Module

class robot.model.totalstatistics.TotalStatistics
    Bases: object
    visit(visitor)
    message

class robot.model.totalstatistics.TotalStatisticsBuilder(suite=None)
    Bases: robot.model.visitor.SuiteVisitor
    add_test(test)
    visit_test(test)
    visit_keyword(kw)
    end_keyword(keyword)
    end_message(msg)
    end_suite(suite)
    end_test(test)
    start_keyword(keyword)
    start_message(msg)
    start_suite(suite)
    start_test(test)
    visit_message(msg)
    visit_suite(suite)

visitor Module

class robot.model.visitor.SuiteVisitor
    Bases: object
    visit_suite(suite)
    start_suite(suite)
    end_suite(suite)
    visit_test(test)
    start_test(test)
    end_test(test)
    visit_keyword(kw)
**start_keyword** *(keyword)*

**end_keyword** *(keyword)*

**visit_message** *(msg)*

**start_message** *(msg)*

**end_message** *(msg)*

```python
class robot.model.visitor.SkipAllVisitor
    Bases: robot.model.visitor.SuiteVisitor

    **visit_suite** *(suite)*
    **visit_keyword** *(kw)*
    **visit_test** *(test)*
    **visit_message** *(msg)*
    **end_keyword** *(keyword)*
    **end_message** *(msg)*
    **end_suite** *(suite)*
    **end_test** *(test)*
```

**output Package**

**output Package**

TImplements runtime logging and listener interface.
This package is likely to change in RF 2.8.

**debugfile Module**

```python
robot.output.debugfile.DebugFile *(path)*
```

**filelogger Module**

class robot.output.filelogger.FileLogger *(path, level)*
    Bases: robot.output.loggerhelper.AbstractLogger

    **message** *(msg)*
    **start_suite** *(suite)*
    **end_suite** *(suite)*
    **start_test** *(test)*
    **end_test** *(test)*

4.1. robot Package
start_keyword (kw)
end_keyword (kw)
output_file (name, path)
close()
debug (msg)
error (msg)
fail (msg)
info (msg)
set_level (level)
trace (msg)
warn (msg)
write (message, level, html=False)

highlighting Module

robot.output.highlighting.Highlighter (stream)
class robot.output.highlighting.AnsiHighlighter (stream)
  Bases: object
    green()
    red()
    yellow()
    reset()
class robot.output.highlighting.NoHighlighting (stream)
  Bases: robot.output.highlighting.AnsiHighlighter
    green()
    red()
    reset()
    yellow()
class robot.output.highlighting.DosHighlighter (stream)
  Bases: object
    green()
    red()
    yellow()
    reset()
listeners Module

```python
class robot.output.listeners.Listeners(listeners):
    Bases: object
    start_suite (*args)
    end_suite (*args)
    start_test (*args)
    end_test (*args)
    start_keyword (*args)
    end_keyword (*args)
    log_message (*args)
    message (*args)
    output_file (*args)
    close (*args)
```

logger Module

```python
class robot.output.logger.Logger
    Bases: robot.output.loggerhelper.AbstractLogger
    A global logger proxy to which new loggers may be registered.

    Whenever something is written to LOGGER in code, all registered loggers are notified. Messages are also
    cached and cached messages written to new loggers when they are registered.

    Tools using Robot Framework’s internal modules should register their own loggers at least to get notifications
    about errors and warnings. A shortcut to get errors/warnings into console is using 'register_console_logger'.

    disable_message_cache()
    disable_automatic_console_logger()
    register_logger (*loggers)
    register_context_changing_logger (logger)
    unregister_logger (*loggers)
    register_console_logger (width=78, colors='AUTO', markers='AUTO', stdout=None, stderr=None)
    register_file_logger (path=None, level='INFO')
    message (msg)
        Messages about what the framework is doing, warnings, errors, ...
    log_message (msg)
        Messages about what the framework is doing, warnings, errors, ...
    log_output (output)
    enable_library_import_logging()
    disable_library_import_logging()
```
output_file (name, path)
    Finished output, report, log, debug, or xunit file

close()
start_suite (suite)
end_suite (suite)
start_test (test)
end_test (test)
start_keyword (keyword)
end_keyword (keyword)
debug (msg)
error (msg)
fail (msg)
info (msg)
set_level (level)
trace (msg)
warn (msg)
write (message, level, html=False)

class robot.output.logger.LoggerCollection
    Bases: object
        register_regular_logger (logger)
        register_context_changing_logger (logger)
        remove_first_regular_logger()
        unregister_logger (logger)
        starting_loggers()
        ending_loggers()
        all_loggers()

loggerhelper Module

class robot.output.loggerhelper.AbstractLogger (level='TRACE')

    set_level (level)
    trace (msg)
    debug (msg)
    info (msg)
    warn (msg)
    fail (msg)
    error (msg)
write (message, level, html=False)

message (msg)

class robot.output.loggerhelper.Message (message, level='INFO', html=False, timestamp=None)

  Bases: robot.model.message.Message
timestamp
message
html
html_message
level
parent
visit (visitor)

class robot.output.loggerhelper.IsLogged (level)

  set_level (level)

class robot.output.loggerhelper.AbstractLoggerProxy (logger)

monitor Module

class robot.output.monitor.CommandLineMonitor (width=78, colors='AUTO', markers='AUTO', stdout=None, stderr=None)

  Bases: object
  start_suite (suite)
  end_suite (suite)
  start_test (test)
  end_test (test)
  start_keyword (kw)
  end_keyword (kw)
  message (msg)
  output_file (name, path)

class robot.output.monitor.CommandLineWriter (width=78, colors='AUTO', markers='AUTO', stdout=None, stderr=None)

  Bases: object
  info (name, doc, start_suite=False)
  suite_separator ()
  test_separator ()
  status (status, clear=False)
  message (message)
  keyword_marker (kw)
  error (message, level, clear=False)
output (name, path)

class robot.output.monitor.StatusHighlighter(colors, *streams)
    Bases: object
    highlight_status (status, stream)
    highlight (text, color, stream)

class robot.output.monitor.KeywordMarker(markers, stdout, highlighter)
    Bases: object
    mark (kw)
    reset_count ()

output Module

class robot.output.output.Output (settings)
    Bases: robot.output.loggerhelper.AbstractLogger
    close (suite)
    start_suite (suite)
    end_suite (suite)
    start_test (test)
    end_test (test)
    start_keyword (kw)
    end_keyword (kw)
    message (msg)
    set_log_level (level)
    debug (msg)
    error (msg)
    fail (msg)
    info (msg)
    set_level (level)
    trace (msg)
    warn (msg)
    write (message, level, html=False)

pyloggingconf Module

robot.output.pyloggingconf.initialize (level)
robot.output.pyloggingconf.set_level (level)

class robot.output.pyloggingconf.RobotHandler (level=0)
    Bases: logging.Handler
    Initializes the instance - basically setting the formatter to None and the filter list to empty.
emit (record)
acquire()
    Acquire the I/O thread lock.
addFilter (filter)
    Add the specified filter to this handler.
close()
    Tidy up any resources used by the handler.
    This version removes the handler from an internal map of handlers, _handlers, which is used for handler lookup by name. Subclasses should ensure that this gets called from overridden close() methods.
createLock()
    Acquire a thread lock for serializing access to the underlying I/O.
filter (record)
    Determine if a record is loggable by consulting all the filters.
    The default is to allow the record to be logged; any filter can veto this and the record is then dropped. Returns a zero value if a record is to be dropped, else non-zero.
flush()
    Ensure all logging output has been flushed.
    This version does nothing and is intended to be implemented by subclasses.
format (record)
    Format the specified record.
    If a formatter is set, use it. Otherwise, use the default formatter for the module.
getName()
handle (record)
    Conditionally emit the specified logging record.
    Emission depends on filters which may have been added to the handler. Wrap the actual emission of the record with acquisition/release of the I/O thread lock. Returns whether the filter passed the record for emission.
handleError (record)
    Handle errors which occur during an emit() call.
    This method should be called from handlers when an exception is encountered during an emit() call. If raiseExceptions is false, exceptions get silently ignored. This is what is mostly wanted for a logging system - most users will not care about errors in the logging system, they are more interested in application errors. You could, however, replace this with a custom handler if you wish. The record which was being processed is passed in to this method.
name
release()
    Release the I/O thread lock.
removeFilter (filter)
    Remove the specified filter from this handler.
setFormatter (fmt)
    Set the formatter for this handler.
setLevel (level)
    Set the logging level of this handler.
set_name(name)

stdoutlogsplitter Module

class robot.output.stdoutlogsplitter.StdoutLogSplitter(output)
    Bases: object
    Splits messages logged through stdout (or stderr) into Message objects

xmllogger Module

class robot.output.xmllogger.XmlLogger(path, log_level='TRACE', generator='Robot')
    Bases: object
    close()
    set_log_level(level)
    message(msg)
    log_message(msg)
    start_keyword(kw)
    end_keyword(kw)
    start_test(test)
    end_test(test)
    start_suite(suite)
    end_suite(suite)
    start_statistics(stats)
    end_statistics(stats)
    start_total_stats(total_stats)
    end_total_stats(total_stats)
    start_tag_stats(tag_stats)
    end_tag_stats(tag_stats)
    start_suite_stats(tag_stats)
    end_suite_stats(tag_stats)
    total_stat(stat)
    suite_stat(stat)
    tag_stat(stat)
    start_errors()
parsing Package

parsing Package

Implements parsing of test data files.

Classes TestCaseFile, TestDataDirectory and ResourceFile represented parsed test data. These can be modified and saved back to disk. In addition, a convenience factory function TestData() can be used to parse file or directory to a corresponding object.

This package is considered stable.

Example

```python
from robot.parsing import TestCaseFile

suite = TestCaseFile(source='path/to/tests.html').populate()
print 'Suite: ', suite.name
for test in suite.testcase_table:
    print test.name

robot.parsing.disable_curdir_processing(method)
```

Decorator to disable processing $CURDIR$ variable.

comments Module

```python
class robot.parsing.comments.CommentCache
    Bases: object
    add(comment)
    consume_with(function)

class robot.parsing.comments.Comments
    Bases: object
    add(row)
    value

class robot.parsing.comments.Comment(comment_data)
    Bases: object
    as_list()
```

datarow Module

```python
class robot.parsing.datarow.DataRow(cells)
    Bases: object
    head
    all
    data
dedent()
handle_old_style_metadata()
```
Robot Framework Documentation, Release 2.7.7

```python
starts_for_loop()

starts_test_or_user_keyword_setting()

test_or_user_keyword_setting_name()

is_indented()

is_continuing()

is_commented()
```

**htmlreader Module**

class robot.parsing.htmlreader.HtmlReader

Bases: HTMLParser.HTMLParser

```
IGNORE = 0
INITIAL = 1
PROCESS = 2

read (htmlfile, populator)

handle_starttag (tag, attrs)

handle_endtag (tag)

handle_data (data)

handle_entityref (name)

handle_charref (number)

handle_pi (data)

unknown_decl (data)

table_start (attrs=None)

table_end ()

tr_start (attrs=None)

tr_end ()

td_start (attrs=None)

td_end ()

br_start (attrs=None)

meta_start (attrs)

CDATA_CONTENT_ELEMENTS = ('script', 'style')

check_for_whole_start_tag (i)

clear_cdata_mode ()

close ()

Handle any buffered data.

entitydefs = None

error (message)```
feed(data)
    Feed data to the parser.
    Call this as often as you want, with as little or as much text as you want (may include ‘n’).

get_starttag_text()
    Return full source of start tag: ‘<...>’.

getpos()
    Return current line number and offset.
goahead(end)
handle_comment(data)
handle_decl(decl)
handle_startendtag(tag, attrs)
parse_bogus_comment(i, report=1)
parse_comment(i, report=1)
parse_declaration(i)
parse_endtag(i)
parse_html_declaration(i)
parse_marked_section(i, report=1)
parse_pi(i)
parse_starttag(i)
reset()
    Reset this instance. Loses all unprocessed data.
set_cdata_mode(elem)
unescape(s)
updatepos(i, j)

model Module

robot.parsing.modelTestData(parent=None, source=None, include_suites=[], warn_on_skipped=False)
    Parses a file or directory to a corresponding model object.

    Parameters
    • parent – (optional) parent to be used in creation of the model object.
    • source – path where test data is read from.

    Returns TestDataSourceDirectory if source is a directory, testCaseFile otherwise.

class robot.parsing.model.TestCaseFile (parent=None, source=None)
    Bases: robot.parsing.model._TestData
    The parsed test case file object.
    populate()
    has_tests()
```python
import
keywords
name

report_invalid_syntax(table, message, level='ERROR')

save(**options)
    Writes this datafile to disk.

    Parameters options – Configuration for writing. These are passed to WritingContext as keyword arguments.

    See also robot.writer.datafilewriter.DataFileWriter

start_table(header_row)

class robot.parsing.model.ResourceFile(source=None)
    Bases: robot.parsing.model._TestData

    The parsed resource file object.

    populate()

    imports

    keywords

    name

    report_invalid_syntax(table, message, level='ERROR')

    save(**options)
    Writes this datafile to disk.

    Parameters options – Configuration for writing. These are passed to WritingContext as keyword arguments.

    See also robot.writer.datafilewriter.DataFileWriter

start_table(header_row)

class robot.parsing.model.TestDataDirectory(parent=None, source=None)
    Bases: robot.parsing.model._TestData

    The parsed test data directory object. Contains hierarchical structure of other TestDataDirectory and TestCaseFile objects.

    populate(include_suites=[], warn_on_skipped=False, recurse=True)

    add_child(path, include_suites)

    has_tests()

    imports

    keywords

    name

    report_invalid_syntax(table, message, level='ERROR')

    save(**options)
    Writes this datafile to disk.

    Parameters options – Configuration for writing. These are passed to WritingContext as keyword arguments.
```
See also `robot.writer.datafilewriter.DataFileWriter`

```python
start_table(header_row)

class robot.parsing.model.TestCaseFileSettingTable(parent)
    Bases: robot.parsing.model._SettingTable
    add_library(name, args=None, comment=None)
    add_metadata(name, value='', comment=None)
    add_resource(name, invalid_args=None, comment=None)
    add_variables(name, args=None, comment=None)
    directory
    get_setter(setting_name)
    header
    is_setting(setting_name)
    name
    normalize(setting)
    report_invalid_syntax(message, level='ERROR')
    set_header(header)
    source
    type = 'setting'

class robot.parsing.model.ResourceFileSettingTable(parent)
    Bases: robot.parsing.model._SettingTable
    add_library(name, args=None, comment=None)
    add_metadata(name, value='', comment=None)
    add_resource(name, invalid_args=None, comment=None)
    add_variables(name, args=None, comment=None)
    directory
    get_setter(setting_name)
    header
    is_setting(setting_name)
    name
    normalize(setting)
    report_invalid_syntax(message, level='ERROR')
    set_header(header)
    source
    type = 'setting'

class robot.parsing.model.InitFileSettingTable(parent)
    Bases: robot.parsing.model._SettingTable
    add_library(name, args=None, comment=None)
```
add_metadata (name, value='', comment=None)
add_resource (name, invalid_args=None, comment=None)
add_variables (name, args=None, comment=None)
directory
get_setter (setting_name)
header
is_setting (setting_name)
name
normalize (setting)
report_invalid_syntax (message, level='ERROR')
set_header (header)
source
type = 'setting'
class robot.parsing.model.VariableTable (parent)
   Bases: robot.parsing.model._Table
type = 'variable'
add (name, value, comment=None)
directory
header
name
report_invalid_syntax (message, level='ERROR')
set_header (header)
source
class robot.parsing.model.TestCaseTable (parent)
   Bases: robot.parsing.model._Table
type = 'test case'
add (name)
is_started ()
directory
header
name
report_invalid_syntax (message, level='ERROR')
set_header (header)
source
class robot.parsing.model.KeywordTable (parent)
   Bases: robot.parsing.model._Table
type = 'keyword'
add(name)
directory
header
name
report_invalid_syntax(message, level='ERROR')
set_header(header)
source
class robot.parsing.model.Variable(name, value, comment=None)
    Bases: object
    as_list()
is_set()
is_for_loop()
has_data()
class robot.parsing.model.TestCase(parent, name)
    Bases: robot.parsing.model._WithSteps, robot.parsing.model._WithSettings
    source
directory
    add_for_loop(declaration, comment=None)
    report_invalid_syntax(message, level='ERROR')
    settings
    add_step(content, comment=None)
copy(name)
    get_setter(setting_name)
    is_setting(setting_name)
    normalize(setting)
class robot.parsing.model.UserKeyword(parent, name)
    Bases: robot.parsing.model.TestCase
    settings
    add_for_loop(declaration, comment=None)
    add_step(content, comment=None)
copy(name)
directory
    get_setter(setting_name)
    is_setting(setting_name)
    normalize(setting)
    report_invalid_syntax(message, level='ERROR')
source

4.1. robot Package
class robot.parsing.model.ForLoop(declaration, comment=None)
    Bases: robot.parsing.model._WithSteps
    is_comment()
    is_for_loop()
    apply_template(template)
    as_list(indent=False, include_comment=True)
    is_set()
    add_step(content, comment=None)
    copy(name)
class robot.parsing.model.Step(content, comment=None)
    Bases: object
    is_comment()
    is_for_loop()
    apply_template(template)
    is_set()
    as_list(indent=False, include_comment=True)
class robot.parsing.model.OldStyleSettingAndVariableTableHeaderMatcher
    Bases: object
    match(header)
class robot.parsing.model.OldStyleTestAndKeywordTableHeaderMatcher
    Bases: object
    match(header)

populators Module

class robot.parsing.populators.FromFilePopulator(datafile)
    Bases: object
    populate(path)
    start_table(header)
    eof()
    add(row)
class robot.parsing.populators.FromDirectoryPopulator
    Bases: object
    ignored_prefixes=('_', '‘')
    ignored_dirs=('CVS',)
    populate(path, datadir, include_suites, warn_on_skipped, recurse=True)

restreader Module

robot.parsing.restreader.RestReader()
settings Module

class robot.parsing.settings.Setting

    setting_name, parent=None, comment=None)

    Bases: object

    reset()

    source

    directory

    populate (value, comment=None)

    Mainly used at parsing time, later attributes can be set directly.

    is_set()

    is_for_loop()

    report_invalid_syntax (message, level='ERROR')

    as_list()

class robot.parsing.settings.StringValueJoiner

    separator

    Bases: object

    join_string_with_value (string, value)

    string_value (value)

class robot.parsing.settings.Documentation

    setting_name, parent=None, comment=None)

    Bases: robot.parsing.settings.Setting

    as_list()

    directory

    is_for_loop()

    is_set()

    populate (value, comment=None)

    Mainly used at parsing time, later attributes can be set directly.

    report_invalid_syntax (message, level='ERROR')

    reset()

    source

class robot.parsing.settings.Template

    setting_name, parent=None, comment=None)

    Bases: robot.parsing.settings.Setting

    is_set()

    as_list()

    directory

    is_for_loop()

    populate (value, comment=None)

    Mainly used at parsing time, later attributes can be set directly.

    report_invalid_syntax (message, level='ERROR')

    reset()

    source
class robot.parsing.settings.Fixture (setting_name, parent=None, comment=None)
    Bases: robot.parsing.settings.Setting
    is_set()
    as_list()
    directory
    is_for_loop()
    populate (value, comment=None)
    Mainly used at parsing time, later attributes can be set directly.
    report_invalid_syntax (message, level='ERROR')
    reset()
    source

class robot.parsing.settings.Timeout (setting_name, parent=None, comment=None)
    Bases: robot.parsing.settings.Setting
    is_set()
    as_list()
    directory
    is_for_loop()
    populate (value, comment=None)
    Mainly used at parsing time, later attributes can be set directly.
    report_invalid_syntax (message, level='ERROR')
    reset()
    source

class robot.parsing.settings.Tags (setting_name, parent=None, comment=None)
    Bases: robot.parsing.settings.Setting
    is_set()
    as_list()
    directory
    is_for_loop()
    populate (value, comment=None)
    Mainly used at parsing time, later attributes can be set directly.
    report_invalid_syntax (message, level='ERROR')
    reset()
    source

class robot.parsing.settings.Arguments (setting_name, parent=None, comment=None)
    Bases: robot.parsing.settings.Setting
    as_list()
    directory
    is_for_loop()
is_set()

populate(value, comment=None)
   Mainly used at parsing time, later attributes can be set directly.

report_invalid_syntax(message, level='ERROR')

reset()

source

class robot.parsing.settings.Return(set_name, parent=None, comment=None)
   Bases: robot.parsing.settings.Setting

   as_list()

directory

is_for_loop()

is_set()

populate(value, comment=None)
   Mainly used at parsing time, later attributes can be set directly.

report_invalid_syntax(message, level='ERROR')

reset()

source

class robot.parsing.settings.Metadata(parent, name, value, comment=None, joined=False)
   Bases: robot.parsing.settings.Setting

   setting_name = 'Metadata'

   reset()

   is_set()

   as_list()

   directory

   is_for_loop()

   populate(value, comment=None)
      Mainly used at parsing time, later attributes can be set directly.

   report_invalid_syntax(message, level='ERROR')

   source

class robot.parsing.settings.Library(parent, name, args=None, alias=None, comment=None)
   Bases: robot.parsing.settings._Import

   as_list()

   directory

   is_for_loop()

   is_set()

   populate(value, comment=None)
      Mainly used at parsing time, later attributes can be set directly.

   report_invalid_syntax(message, level='ERROR')
reset()
source
type
class robot.parsing.settings.Resource (parent, name, invalid_args=None, comment=None)
    Bases: robot.parsing.settings._Import
    as_list()
directory
    is_for_loop()
    is_set()
    populate(value, comment=None)
        Mainly used at parsing time, later attributes can be set directly.
    report_invalid_syntax(message, level='ERROR')
reset()
source
type
class robot.parsing.settings.Variables (parent, name, args=None, comment=None)
    Bases: robot.parsing.settings._Import
    as_list()
directory
    is_for_loop()
    is_set()
    populate(value, comment=None)
        Mainly used at parsing time, later attributes can be set directly.
    report_invalid_syntax(message, level='ERROR')
reset()
source
type
class robot.parsing.settings.ImportList (parent)
    Bases: robot.parsing.settings._DataList
    populate_library(data, comment)
    populate_resource(data, comment)
    populate_variables(data, comment)
    add(meta)
class robot.parsing.settings.MetadataList (parent)
    Bases: robot.parsing.settings._DataList
    populate(name, value, comment)
    add(meta)
tablepopulators Module

class robot.parsing.tablepopulators.Populator
    Bases: object
    Explicit interface for all populators.
    add(row)
    populate()

class robot.parsing.tablepopulators.NullPopulator
    Bases: robot.parsing.tablepopulators.Populator
    add(row)
    populate()

class robot.parsing.tablepopulators.SettingTablePopulator(table)
    Bases: robot.parsing.tablepopulators._TablePopulator
    add(row)
    populate()

class robot.parsing.tablepopulators.VariableTablePopulator(table)
    Bases: robot.parsing.tablepopulators._TablePopulator
    populate()
    add(row)

class robot.parsing.tablepopulators.TestTablePopulator(table)
    Bases: robot.parsing.tablepopulators._StepContainingTablePopulator
    add(row)
    populate()

class robot.parsing.tablepopulators.KeywordTablePopulator(table)
    Bases: robot.parsing.tablepopulators._StepContainingTablePopulator
    add(row)
    populate()

class robot.parsing.tablepopulators.ForLoopPopulator(for_loop_creator)
    Bases: robot.parsing.tablepopulators.Populator
    add(row)
    populate()

class robot.parsing.tablepopulators_TestCasePopulator(test_or_uk_creator)
    Bases: robot.parsing.tablepopulators._TestCaseUserKeywordPopulator
    add(row)
    populate()

class robot.parsing.tablepopulators.UserKeywordPopulator(test_or_uk_creator)
    Bases: robot.parsing.tablepopulators._TestCaseUserKeywordPopulator
    add(row)
    populate()
class robot.parsing.tablepopulators.VariablePopulator(setter, name)
    Bases: robot.parsing.tablepopulators._PropertyPopulator
    populate()
    add(row)

class robot.parsing.tablepopulators.SettingPopulator(setter)
    Bases: robot.parsing.tablepopulators._PropertyPopulator
    populate()
    add(row)

class robot.parsing.tablepopulators.DocumentationPopulator(setter)
    Bases: robot.parsing.tablepopulators._PropertyPopulator
    populate()
    add(row)

class robot.parsing.tablepopulators.MetadataPopulator(setter)
    Bases: robot.parsing.tablepopulators.DocumentationPopulator
    populate()
    add(row)

class robot.parsing.tablepopulators.StepPopulator(setter)
    Bases: robot.parsing.tablepopulators._PropertyPopulator
    populate()
    add(row)

tsvreader Module

class robot.parsing.tsvreader.TsvReader
    read(tsvfile, populator)
    classmethod split_row(row)

txtreader Module

class robot.parsing.txtreader.TxtReader
    Bases: robot.parsing.tsvreader.TsvReader
    classmethod split_row(row)
    read(tsvfile, populator)

reporting Package

reporting Package

Implements report and log file generation.
This package is considered stable.
**jsbuildingcontext Module**

```python
class robot.reporting.jsbuildingcontext.JsBuildingContext (log_path=None, split_log=False, prune_input=False):
```

- `string (string, escape=True)`
- `html (string)`
- `relative_source (source)`
- `timestamp (time)`
- `message_level (level)`
- `create_link_target (msg)`
- `link (msg)`
- `strings`
- `startSplittingIfNeeded (split=False)`
- `endSplitting (model)`
- `prune_input (*args, **kwds)`

**jsexecutionresult Module**

```python
class robot.reporting.jsexecutionresult.JsExecutionResult (suite, statistics, errors, strings, basemillis=None, split_results=None, min_level=None):
```

- `remove_data_not_needed_in_report()`

**jsmodelbuilders Module**

```python
class robot.reporting.jsmodelbuilders.JsModelBuilder (log_path=None, split_log=False, prune_input_to_save_memory=False):
```

- `build_from (result_from_xml)`

```python
class robot.reporting.jsmodelbuilders.SuiteBuilder (context):
    Bases: robot.reporting.jsmodelbuilders._Builder
    build (suite)
```

```python
class robot.reporting.jsmodelbuilders.TestBuilder (context):
    Bases: robot.reporting.jsmodelbuilders._Builder
    build (test)
```

```python
class robot.reporting.jsmodelbuilders.KeywordBuilder (context):
    Bases: robot.reporting.jsmodelbuilders._Builder
    build (kw, split=False)
```
class robot.reporting.jsmodelbuilders.MessageBuilder(context)
    Bases: robot.reporting.jsmodelbuilders._Builder

    build(msg)

class robot.reporting.jsmodelbuilders.StatisticsBuilder
    Bases: object

    build(statistics)

class robot.reporting.jsmodelbuilders.ErrorsBuilder(context)
    Bases: robot.reporting.jsmodelbuilders._Builder

    build(errors)

class robot.reporting.jsmodelbuilders.ErrorMessageBuilder(context)
    Bases: robot.reporting.jsmodelbuilders.MessageBuilder

    build(msg)

jswriter Module

class robot.reporting.jswriter.JsResultWriter(output, start_block='&lt;script
    type="text/javascript"&gt;n',
    end_block='&lt;/script&gt;n',
    split_threshold=9500)

    Bases: object

    write(result, settings)

class robot.reporting.jswriter.SuiteWriter(write_json, split_threshold)
    Bases: object

    write(suite, variable)

class robot.reporting.jswriter.SplitLogWriter(output)
    Bases: object

    write(keywords, strings, index, notify)

logreportwriters Module

class robot.reporting.logreportwriters.LogWriter(js_model)
    Bases: robot.reporting.logreportwriters._LogReportWriter

    write(path, config)

class robot.reporting.logreportwriters.ReportWriter(js_model)
    Bases: robot.reporting.logreportwriters._LogReportWriter

    write(path, config)

class robot.reporting.logreportwriters.RobotModelWriter(output, model, config)
    Bases: robot.htmldata.htmlfilewriter.ModelWriter

    write(line)

    handles(line)
outputwriter Module

class robot.reporting.outputwriter.OutputWriter(output)
    Bases: robot.output.xmllogger.XmlLogger, robot.result.visitor.ResultVisitor

    start_message(msg)
    close()
    start_errors(errors)
    end_errors(errors)
    end_result(result)
    start_total_statistics(total_stats)
    start_tag_statistics(tag_stats)
    start_suite_statistics(tag_stats)
    end_total_statistics(total_stats)
    end_tag_statistics(tag_stats)
    end_suite_statistics(tag_stats)
    visit_stat(stat)
    end_keyword(kw)
    end_message(msg)
    end_stat(stat)
    end_statistics(stats)
    end_suite(suite)
    end_suite_stats(tag_stats)
    end_tag_stats(tag_stats)
    end_test(test)
    end_total_stats(total_stats)
    log_message(msg)
    message(msg)
    set_log_level(level)
    start_keyword(kw)
    start_result(result)
    start_stat(stat)
    start_statistics(stats)
    start_suite(suite)
    start_suite_stats(tag_stats)
    start_tag_stats(tag_stats)
    start_test(test)
    start_total_stats(total_stats)
suite_stat (stat)
tag_stat (stat)
total_stat (stat)
visit_errors (errors)
visit_keyword (kw)
visit_message (msg)
visit_result (result)
visit_statistics (stats)
visit_suite (suite)
visit_suite_statistics (stats)
visit_tag_statistics (stats)
visit_test (test)
visit_total_statistics (stats)

resultwriter Module

class robot.reporting.resultwriter.ResultWriter (*data_sources)
    Bases: object
    write_results (settings, results=None)

class robot.reporting.resultwriter.Results (data_sources, settings)
    Bases: object
    result
    js_result

stringcache Module

class robot.reporting.stringcache.StringIndex
    Bases: long
    bit_length () → int or long
        Number of bits necessary to represent self in binary. >>> bin(37L) ‘0b100101’ >>> (37L).bit_length() 6
    conjugate ()
        Returns self, the complex conjugate of any long.
    denominator
        the denominator of a rational number in lowest terms
    imag
        the imaginary part of a complex number
    numerator
        the numerator of a rational number in lowest terms
    real
        the real part of a complex number
class robot.reporting.stringcache.StringCache
    Bases: object
    
    add(text)
    dump()

xunitwriter Module

class robot.reporting.xunitwriter.XUnitWriter(execution_result)
    Bases: object
    
    write(output)

class robot.reporting.xunitwriter.XUnitFileWriter(xml_writer)
    Bases: robot.result.visitor.ResultVisitor

    Provides an xUnit-compatible result file.
    Attempts to adhere to the de facto schema guessed by Peter Reilly, see: http://marc.info/?l=ant-dev&m=123551933508682
    
    start_suite(suite)
    end_suite(suite)
    start_test(test)
    end_test(test)
    visit_keyword(kw)
    end_result(result)
    end_errors(errors)
    end_keyword(keyword)
    end_message(msg)
    end_stat(stat)
    end_statistics(stats)
    end_suite_statistics(suite_stats)
    end_tag_statistics(stats)
    end_total_statistics(stats)
    start_errors(errors)
    start_keyword(keyword)
    start_message(msg)
    start_result(result)
    start_stat(stat)
    start_statistics(stats)
    start_suite_statistics(stats)
    start_tag_statistics(stats)
    start_total_statistics(stats)
visit_errors (errors)
visit_message (msg)
visit_result (result)
visit_stat (stat)
visit_statistics (stats)
visit_suite (suite)
visit_suite_statistics (stats)
visit_tag_statistics (stats)
visit_test (test)
visit_total_statistics (stats)

class robot.reporting.xunitwriter.TestFailureWriter (writer)
Bases: robot.result.visitor.ResultVisitor

start_test (test)
end_test (test)

visit_message (msg)
    Populates the <failure> section, normally only with a ‘Stacktrace’.
    There is a weakness here because filtering is based on message level: - DEBUG level is used by RF for
    ‘Tracebacks’ (what is expected here) - INFO and TRACE are used for keywords and arguments (not errors)
    - first FAIL message is already reported as <failure> attribute
end_errors (errors)
end_keyword (keyword)
end_message (msg)
end_result (result)
end_stat (stat)
end_statistics (stats)
end_suite (suite)
end_suite_statistics (suite_stats)
end_tag_statistics (stats)
end_total_statistics (stats)
start_errors (errors)
start_keyword (keyword)
start_message (msg)
start_result (result)
start_stat (stat)
start_statistics (stats)
start_suite (suite)
start_suite_statistics (stats)
result Package

result Package

Implements parsing results from XML output files.
The entry point of this API is the `ExecutionResult()` factory method, which returns an instance of `Result`.
This package is considered stable.

Example    The example below reads a given output file and marks each test case whose execution time is longer than three minutes failed. The `Result` object is then written back to disk and normal log and report files could be generated with `rebot` tool.

```
#!/usr/bin/env python

"""Usage: check_test_times.py inpath [outpath]

Reads result of a test run from Robot output file and checks that no test took longer than 3 minute to execute. If outpath is not given, the result is written over the original file.
"""

import sys
from robot.result import ExecutionResult

def check_tests(inpath, outpath=None):
    if not outpath:
        outpath = inpath
    result = ExecutionResult(inpath)
    _check_execution_times(result.suite)
    result.save(outpath)

def _check_execution_times(suite):
    for test in suite.tests:
        if test.status == 'PASS' and test.elapsedtime > 1000 * 60 * 3:
test.status = 'FAIL'
test.message = 'Test execution time was too long: %s' % test.elapsedtime
for suite in suite.suites:
    _check_execution_times(suite)

if __name__ == '__main__':
    try:
        check_tests(*sys.argv[1:])
    except TypeError:
        print __doc__

configurer Module

class robot.result.configurer.SuiteConfigurer (name=None, doc=None, metadata=None, set_tags=None, include_tags=None, exclude_tags=None, include_suites=None, include_tests=None, process_empty_suite=False, remove_keywords=None, log_level=None, critical=None, noncritical=None, starttime=None, endtime=None)

    Bases: object
    add_tags
    remove_tags
    configure(suite)

executionerrors Module

class robot.result.executionerrors.ExecutionErrors (messages=None)
    Bases: object
    message_class
        alias of Message
    messages
    add(other)
    visit(visitor)

executionresult Module

class robot.result.executionresult.Result (source=None, root_suite=None, errors=None)
    Bases: object

    Contains results of test execution.

    Variables

    • source – Path to the xml file where results are read from.
    • suite – Hierarchical TestSuite results.
    • errors – Execution ExecutionErrors.
**statistics**

Test execution **Statistics**.

**return_code**

Return code (integer) of test execution.

**configure** *(status_rc=True, suite_config={}, stat_config={})*

**visit** *(visitor)*

**save** *(path=None)*

class **robot.result.executionresult.**CombinedResult**(others)**

Bases: **robot.result.executionresult.**Result

**add_result** *(other)*

**configure** *(status_rc=True, suite_config={}, stat_config={})*

**return_code**

Return code (integer) of test execution.

**save** *(path=None)*

**statistics**

Test execution **Statistics**.

**visit** *(visitor)*

---

**keyword Module**

class **robot.result.keyword.**Keyword**(name='', doc='', args=None, type='kw', timeout='', status='FAIL', starttime=None, endtime=None)**

Bases: **robot.model.keyword.**Keyword

Results of a single keyword.

**Variables**

- **name** – Keyword name.
- **parent** – **TestSuite** or **TestCase** that contains this keyword.
- **doc** – Keyword documentation.
- **args** – Keyword arguments, a list of strings.
- **type** – ‘SETUP’, ‘TEARDOWN’ or ‘KW’.
- **timeout** – Keyword timeout.
- **messages** – Log messages, a list of **Message** instances.
- **keywords** – Child keyword results, a list of **Keyword** instances
- **status** – String ‘PASS’ of ‘FAIL’.
- **starttime** – Keyword execution start time as a timestamp.
- **endtime** – Keyword execution end time as a timestamp.

**message_class**

alias of **Message**

**status**

**starttime**
endtime
message
elapsedtime
passed
FOR_ITEM_TYPE = ‘foritem’
FOR_LOOP_TYPE = ‘for’
KEYWORD_TYPE = ‘kw’
SETUP_TYPE = ‘setup’
TEARDOWN_TYPE = ‘teardown’
args
doc
id
keywords
messages
name
parent
timeout
type
visit(visitor)

**keywordremover Module**

robot.result.keywordremover.KeywordRemover(*how*)
class robot.result.keywordremover.AllKeywordsRemover
    Bases: robot.result.keywordremover._KeywordRemover
    
visit_keyword(keyword)
end_keyword(keyword)
end_message(msg)
end_suite(suite)
end_test(test)
start_keyword(keyword)
start_message(msg)
start_suite(suite)
start_test(test)
visit_message(msg)
visit_suite(suite)
visit_test(test)
class robot.result.keywordremover.PassedKeywordRemover
    Bases: robot.result.keywordremover._KeywordRemover
    start_suite (suite)
    visit_test (test)
    visit_keyword (keyword)
    end_keyword (keyword)
    end_message (msg)
    end_suite (suite)
    end_test (test)
    start_keyword (keyword)
    start_message (msg)
    start_test (test)
    visit_message (msg)
    visit_suite (suite)

class robot.result.keywordremover.ForLoopItemsRemover
    Bases: robot.result.keywordremover._KeywordRemover
    start_keyword (kw)
    end_keyword (keyword)
    end_message (msg)
    end_suite (suite)
    end_test (test)
    start_message (msg)
    start_suite (suite)
    start_test (test)
    visit_keyword (kw)
    visit_message (msg)
    visit_suite (suite)
    visit_test (test)

class robot.result.keywordremover.WaitUntilKeywordSucceedsRemover
    Bases: robot.result.keywordremover._KeywordRemover
    start_keyword (kw)
    end_keyword (keyword)
    end_message (msg)
    end_suite (suite)
    end_test (test)
    start_message (msg)
    start_suite (suite)
start_test (test)
visit_keyword (kw)
visit_message (msg)
visit_suite (suite)
visit_test (test)
class robot.result.keywordremover.ContainsWarning
    Bases: robot.model.visitor.SuiteVisitor
    start_suite (suite)
    start_test (test)
    start_keyword (keyword)
    visit_message (msg)
    end_keyword (keyword)
    end_message (msg)
    end_suite (suite)
    end_test (test)
    start_message (msg)
    visit_keyword (kw)
    visit_suite (suite)
    visit_test (test)
class robot.result.keywordremover.RemovalMessage (message)
    Bases: object
    set_if_removed (kw, len_before)
    set (kw, message=None)

message Module
class robot.result.message.Message (message='', level='INFO', html=False, timestamp=None, parent=None)
    Bases: robot.model.message.Message
    html
    html_message
    level
    message
    parent
    timestamp
    visit (visitor)
messagefilter Module

class robot.result.messagefilter.MessageFilter(loglevel)
    Bases: robot.model.visitor.SuiteVisitor

    start_keyword(keyword)
    end_keyword(keyword)
    end_message(msg)
    end_suite(suite)
    end_test(test)
    start_message(msg)
    start_suite(suite)
    start_test(test)
    visit_keyword(kw)
    visit_message(msg)
    visit_suite(suite)
    visit_test(test)

resultbuilder Module

robot.result.resultbuilder.ExecutionResult(*sources, **options)
Constructs Result object based on execution result xml file(s).

    Parameters

        • sources – The Robot Framework output xml file(s).
        • options – Configuration options passed to ExecutionResultBuilder as keyword arguments. New in 2.7.5.

    Returns Result instance.

    See robot.result for usage example.

class robot.result.resultbuilder.ExecutionResultBuilder(source, include_keywords=True)
    Bases: object

    build(result)

suiteteardownfailed Module

class robot.result.suiteteardownfailed.SuiteTeardownFailureHandler(suite_generator)
    Bases: robot.model.visitor.SuiteVisitor

    start_suite(suite)
    end_suite(suite)
    visit_test(test)
    visit_keyword(keyword)
end_keyword(keyword)
end_message(msg)
end_test(test)
start_keyword(keyword)
start_message(msg)
start_test(test)
visit_message(msg)
visit_suite(suite)

class robot.result.suitetearndownfailed.SuiteTeardownFailed(error)
    Bases: robot.model.visitor.SuiteVisitor
    visit_test(test)
    visit_keyword(keyword)
    end_keyword(keyword)
    end_message(msg)
    end_suite(suite)
    end_test(test)
    start_keyword(keyword)
    start_message(msg)
    start_suite(suite)
    start_test(test)
    visit_message(msg)
    visit_suite(suite)

testcase Module

class robot.result.testcase.TestCase(name='', doc='', tags=\None, timeout='', status='FAIL', message='', starttime=None, endtime=None)
    Bases: robot.model.testcase.TestCase

Results of a single test case.

Variables

• name – Test case name.
• parent – TestSuite that contains this test.
• doc – Test case documentation.
• tags – Test case tags, a list of strings.
• timeout – Test case timeout.
• keywords – Keyword results, a list of Keyword instances and contains also possible setup and teardown keywords.
• status – String ‘PASS’ of ‘FAIL’.
• **message** – Possible failure message.
• **starttime** – Test case execution start time as a timestamp.
• **endtime** – Test case execution end time as a timestamp.

```python
keyword_class
    alias of Keyword

status
message
starttime
endtime
elapsedtime
passed
critical
doc
id
keywords
longname
name
parent
tags
timeout
visit(visitor)
```

testsuite Module

class robot.result.testsuite.TestSuite(source='', name='', doc='', metadata=None, message='', starttime=None, endtime=None)
Bases: robot.model.testsuite.TestSuite

Results of a single test suite.

Variables

• **parent** – Parent TestSuite or None.
• **source** – Path to the source file.
• **name** – Test suite name.
• **doc** – Test suite documentation.
• **metadata** – Test suite metadata as a dictionary.
• **suites** – Child suite results.
• **tests** – Test case results. a list of TestCase instances.
• **keywords** – A list containing setup and teardown results.
• **message** – Possible failure message.
• **starttime** – Test suite execution start time as a timestamp.

• **endtime** – Test suite execution end time as a timestamp.

test_class
  alias of TestCase

keyword_class
  alias of Keyword

message

starttime

document

endtime

status

statistics

document

full_message

elapsedtime

remove_keywords (how)

filter_messages (log_level='TRACE')

criticality

document

filter (included_suites=None, included_tests=None, included_tags=None, excluded_tags=None)

id

keywords

longname

metadata

name

parent

set_criticality (critical_tags=None, non_critical_tags=None)

set_tags (add=None, remove=None)

source

suites

tests

test_count

visit (visitor)

visitor Module

class robot.result.visitor.ResultVisitor
  Bases: robot.model.visitor.SuiteVisitor

  visit_result (result)

  start_result (result)
end_result (result)
visit_statistics (stats)
start_statistics (stats)
end_statistics (stats)
visit_total_statistics (stats)
start_total_statistics (stats)
end_total_statistics (stats)
visit_tag_statistics (stats)
start_tag_statistics (stats)
end_tag_statistics (stats)
visit_suite_statistics (stats)
start_suite_statistics (stats)
end_suite_statistics (stats)
visit_stat (stat)
start_stat (stat)
end_stat (stat)
visit_errors (errors)
start_errors (errors)
end_errors (errors)
end_keyword (keyword)
end_message (msg)
end_suite (suite)
end_test (test)
start_keyword (keyword)
start_message (msg)
start_suite (suite)
start_test (test)
visit_keyword (kw)
visit_message (msg)
visit_suite (suite)
visit_test (test)

xmlelementhandlers Module

class robot.result.xmlelementhandlers.XmlElementHandler (execution_result,
root_handler=None)

Bases: object
start (elem)
class robot.result.xmlelementhandlers.RootHandler
    Bases: robot.result.xmlelementhandlers._Handler
    end(elem, result)
    handle_child(elem, result)
    start(elem, result)

class robot.result.xmlelementhandlers.RobotHandler
    Bases: robot.result.xmlelementhandlers._Handler
    tag = 'robot'
    start(elem, result)
    end(elem, result)
    handle_child(elem, result)

class robot.result.xmlelementhandlers.SuiteHandler
    Bases: robot.result.xmlelementhandlers._Handler
    tag = 'suite'
    start(elem, result)
    end(elem, result)
    handle_child(elem, result)

class robot.result.xmlelementhandlers.RootSuiteHandler
    Bases: robot.result.xmlelementhandlers.SuiteHandler
    start(elem, result)
    end(elem, result)
    handle_child(elem, result)

class robot.result.xmlelementhandlers.TestCaseHandler
    Bases: robot.result.xmlelementhandlers._Handler
    tag = 'test'
    start(elem, result)
    end(elem, result)
    handle_child(elem, result)

class robot.result.xmlelementhandlers.KeywordHandler
    Bases: robot.result.xmlelementhandlers._Handler
    tag = 'kw'
    start(elem, result)
    end(elem, result)
    handle_child(elem, result)

class robot.result.xmlelementhandlers.MessageHandler
    Bases: robot.result.xmlelementhandlers._Handler
tag = 'msg'
end(elem, result)
handle_child(elem, result)
start(elem, result)
class robot.result.xmlelementhandlers.KeywordStatusHandler
    Bases: robot.result.xmlelementhandlers._StatusHandler
    end(elem, result)
    handle_child(elem, result)
    start(elem, result)
tag = 'status'
class robot.result.xmlelementhandlers_SUITEStatusHandler
    Bases: robot.result.xmlelementhandlers._StatusHandler
    end(elem, result)
    handle_child(elem, result)
    start(elem, result)
tag = 'status'
class robot.result.xmlelementhandlers_TestCaseStatusHandler
    Bases: robot.result.xmlelementhandlers._StatusHandler
    end(elem, result)
    handle_child(elem, result)
    start(elem, result)
tag = 'status'
class robot.result.xmlelementhandlers.DocHandler
    Bases: robot.result.xmlelementhandlers._Handler
    tag = 'doc'
    end(elem, result)
    handle_child(elem, result)
    start(elem, result)
class robot.result.xmlelementhandlers.MetadataHandler
    Bases: robot.result.xmlelementhandlers._Handler
    tag = 'metadata'
    end(elem, result)
    handle_child(elem, result)
    start(elem, result)
class robot.result.xmlelementhandlers.MetadataItemHandler
    Bases: robot.result.xmlelementhandlers._Handler
    tag = 'item'
    end(elem, result)
handle_child(elem, result)
start(elem, result)

class robot.result.xmlelementhandlers.TagsHandler
    Bases: robot.result.xmlelementhandlers._Handler
    tag = 'tags'
    end(elem, result)
    handle_child(elem, result)
    start(elem, result)

class robot.result.xmlelementhandlers.TagHandler
    Bases: robot.result.xmlelementhandlers._Handler
    tag = 'tag'
    end(elem, result)
    handle_child(elem, result)
    start(elem, result)

class robot.result.xmlelementhandlers.ArgumentsHandler
    Bases: robot.result.xmlelementhandlers._Handler
    tag = 'arguments'
    end(elem, result)
    handle_child(elem, result)
    start(elem, result)

class robot.result.xmlelementhandlers.ArgumentHandler
    Bases: robot.result.xmlelementhandlers._Handler
    tag = 'arg'
    end(elem, result)
    handle_child(elem, result)
    start(elem, result)

class robot.result.xmlelementhandlers.ErrorsHandler
    Bases: robot.result.xmlelementhandlers._Handler
    tag = 'errors'
    start(elem, result)
    end(elem, result)
    handle_child(elem, result)

class robot.result.xmlelementhandlers.StatisticsHandler
    Bases: robot.result.xmlelementhandlers._Handler
    tag = 'statistics'
    handle_child(elem, result)
    end(elem, result)
    start(elem, result)
**running Package**

Implements the core test execution logic.

The code in this package is in many places suboptimal and likely to change in RF 2.8. External code should use this package with care.

Currently, the main entry point is the `TestSuite()` factory method.

```python
robot.running.UserLibrary(path)
```

Create a user library instance from given resource file.

This is used at least by libdoc.py.

**arguments Module**

```python
class robot.running.arguments.PythonKeywordArguments:
    Bases: robot.running.arguments._KeywordArguments
    argument_source, kw_or_lib_name
    check_arg_limits(args, namedargs=())
    check_arg_limits_for_dry_run(args)
    resolve(args, variables, output=None)

class robot.running.arguments.JavaKeywordArguments:
    Bases: robot.running.arguments._KeywordArguments
    handler_method, name
    check_arg_limits(args, namedargs=())
    check_arg_limits_for_dry_run(args)
    resolve(args, variables, output=None)

class robot.running.arguments.DynamicKeywordArguments:
    Bases: robot.running.arguments._KeywordArguments
    argument_source, kw_or_lib_name
    check_arg_limits(args, namedargs=())
    check_arg_limits_for_dry_run(args)
    resolve(args, variables, output=None)

class robot.running.arguments.RunKeywordArguments:
    Bases: robot.running.arguments.PythonKeywordArguments
    argument_source, name, arg_resolution_index
    check_arg_limits(args, namedargs=())
    check_arg_limits_for_dry_run(args)
    resolve(args, variables, output=None)

class robot.running.arguments.PythonInitArguments:
    Bases: robot.running.arguments.PythonKeywordArguments
    argument_source, kw_or_lib_name
    check_arg_limits(args, namedargs=())
    check_arg_limits_for_dry_run(args)
    resolve(args, variables, output=None)
```
class robot.running.arguments.JavaInitArguments (handler_method, name)
    Bases: robot.running.arguments.JavaKeywordArguments
    resolve (args, variables=None)
    check_arg_limits (args, namedargs={})
    check_arg_limits_for_dry_run (args)

class robot.running.arguments.UserKeywordArguments (args, name)
    Bases: object
    resolve_arguments_for_dry_run (arguments)
    resolve (arguments, variables, output)
    set_variables (arg_values, variables, output)

class robot.running.arguments.UserKeywordArgsTemplate (minargs, defaults)
    Bases: object
    set_value (idx, value)
    as_list ()

class robot.running.arguments.UserKeywordArgumentResolver (arguments)
    Bases: robot.running.arguments._ArgumentResolver
    resolve (values, output, variables=None)

class robot.running.arguments.PythonKeywordArgumentResolver (arguments)
    Bases: robot.running.arguments._ArgumentResolver
    resolve (values, output, variables=None)

class robot.running.arguments.JavaKeywordArgumentResolver (arguments)
    Bases: object
    resolve (values, output, variables)

class robot.running.context.ExecutionContexts
    Bases: object
    current
    top
    namespaces
    start_suite (namespace, output, dry_run=False)
    end_suite ()

defaultvalues Module

class robot.running.defaultvalues.DefaultValues (settings, parent_default_values=None)
    Bases: object
    get_setup (tc_setup)
    get_teardown (tc_teardown)
get_timeout(tc_timeout)
get_tags(tc_tags)
get_template(template)

fixture Module

class robot.running.fixture.Setup(name, args)
    Bases: robot.running.fixture._Fixture
    replace_variables(variables, errors)
    run(context)
    serialize(serializer)

class robot.running.fixture.Teardown(name, args)
    Bases: robot.running.fixture._Fixture
    replace_variables(variables, errors)
    run(context)
    serialize(serializer)

handlers Module

robot.running.handlers.Handler(library, name, method)
robot.running.handlers.DynamicHandler(library, name, method, doc, argspec)
robot.running.handlers.InitHandler(library, method, docgetter=None)

importer Module

class robot.running.importer.Importer
    Bases: object
    reset()
    import_library(name, args=None, alias=None, variables=None)
    import_resource(path)

class robot.running.importer.ImportCache
    Keeps track on and optionally caches imported items.
    Handles paths in keys case-insensitively on case-insensitive OSes. Unlike dicts, this storage accepts mutable values in keys.
    add(key, item=None)
    values()
javaargcoercer Module

keywords Module

class robot.running.keywords.Keywords(steps, template=None)
    Bases: object
        add_keyword(keyword)
        run(context)

class robot.running.keywords.Keyword(name, args, assign=None, type='kw')
    Bases: robot.common.keyword.BaseKeyword
        run(context)
        passed
        serialize(serializer)

class robot.running.keywords.ForLoop(forstep, template=None)
    Bases: robot.common.keyword.BaseKeyword
        run(context)
        passed
        serialize(serializer)

model Module

robot.running.model.TestSuite(datasources, settings, process_variables=True)
    Creates a runnable test suite from given data sources and settings.
    This is a factory method that returns either RunnableTestSuite or RunnableMultiTestSuite depending on one or more data sources given. This method, and especially the returned suite, is likely to change heavily in version 2.8.

    Parameters
        • datasources – List of paths to read data from. Starting from 2.7.2, a single datasource can also be given as a string.
        • settings(RobotSettings) – Execution configuration.

    Returns RunnableTestSuite

class robot.running.modelRunnableTestSuite(data, parent=None, defaults=None, process_variables=True)
    Bases: robot.common.model.BaseTestSuite
        filter_emptySuites()
        run(output, parent_context=None, errors=None)
        filter(suites=None, tests=None, includes=None, excludes=None, zero_tests_ok=False)
        filter_by_names(suites=None, tests=None, zero_tests_ok=False)
        filter_by_tags(includes=None, excludes=None, zero_tests_ok=False)
        get_full_message()
            Returns suite’s message including statistics message
get_stat_message()
get_test_count()
id
longname
name
return_code
serialize(serializer)
set_critical_tags(critical, non_critical)
set_doc(doc)
set_metadata(metalist)
set_name(name)
set_options(settings)
set_runmode(runmode)
set_status()
    Sets status and statistics based on subsuite and test statuses.
    Can/should be used when statuses have been changed somehow.
set_tags(tags)
suite_teardown_failed(error=None, message=None)
class robot.running.modelRunnableMultiTestSuite(suitedatas, process_variables=True)
    Bases: robot.running.model.RunnableTestSuite
    filter(suites=None, tests=None, includes=None, excludes=None, zero_tests_ok=False)
    filter_by_names(suites=None, tests=None, zero_tests_ok=False)
    filter_by_tags(includes=None, excludes=None, zero_tests_ok=False)
    filter_emptySuites()
    get_full_message()
        Returns suite’s message including statistics message
    get_stat_message()
    get_test_count()
id
longname
name
return_code
run(output, parent_context=None, errors=None)
serialize(serializer)
set_critical_tags(critical, non_critical)
set_doc(doc)
set_metadata(metalist)
set_name(name)
set_options(settings)
set_runmode(runmode)
set_status()
Sets status and statistics based on subsuite and test statuses.
Can/should be used when statuses have been changed somehow.
set_tags(tags)
suite_teardown_failed(error=None, message=None)

class robot.running.model.RunnableTestCase(tc_data, parent, defaults)
    Bases: robot.common.model.BaseTestCase
    run(context, parent_errors)
    id
    is_included(incl_tags, excl_tags)
        Returns True if this test case is included but not excluded.
        If no ‘incl_tags’ are given all tests are considered to be included.
    longname
    passed
    serialize(serializer)
    set_criticality(critical)
    suite_teardown_failed(message)

namespace Module

class robot.running.namespace.Namespace(suite, parent_vars)

    handle_imports()
    import_resource(name, overwrite=True)
    import_variables(name, args, overwrite=False, variables=None)
    import_library(name, args=None, alias=None, variables=None)
    start_test(test)
    end_test()
    set_test_status_before_teardown(message, status)
    end_suite()
    start_user_keyword(handler)
    end_user_keyword()
    get_library_instance(libname)
    get_handler(name)
outputcapture Module

class robot.running.outputcapture.OutputCapturer (library_import=False)

    release_and_log()

runerrors Module

class robot.running.runerrors.SuiteRunErrors (exit_on_failure_mode=False, skip_teardowns_on_exit_mode=False)

    Bases: object
    exit
    start_suite()
    end_suite()
    is_setup_allowed()
    is_teardown_allowed()
    suite_initialized(error=None)
    setup_executed(error=None)
    get_suite_error()
    get_child_error()
    test_failed(exit=False, critical=False)

class robot.running.runerrors.TestRunErrors (parent)

    Bases: object

    is_run_allowed()
    is_teardown_allowed()
    test_initialized(error=None)
    setup_failed(error)
    keyword_failed(error)
    teardown_failed(error)
    test_failed(exit=False, critical=False)
    get_message()
    get_teardown_message(message)
    get_parent_or_init_error()
runkwregister Module

signalhandler Module

testlibraries Module

robot.running.testlibraries.\texttt{TestLibrary}(name, \texttt{args=\texttt{None}}, \texttt{variables=\texttt{None}}, \texttt{create_handlers=\texttt{True}})

userkeyword Module

class robot.running.userkeyword.\texttt{UserLibrary}(user_keywords, path=\texttt{None})
    Bases: robot.common.libraries.BaseLibrary
        \texttt{supports\_named\_arguments} = \texttt{True}
        \texttt{has\_handler}(name)
        \texttt{get\_handler}(name)

class robot.running.userkeyword.\texttt{UserKeywordHandler}(keyword, libname)
    Bases: object
        \texttt{type} = \texttt{\textquote{user}}
        \texttt{longname}
        \texttt{shortdoc}
        \texttt{init\_keyword}(varz)
        \texttt{run}(context, arguments)

class robot.running.userkeyword.\texttt{EmbeddedArgsTemplate}(keyword, libname)
    Bases: robot.running.userkeyword.UserKeywordHandler
        \texttt{init\_keyword}(varz)
        \texttt{longname}
        \texttt{run}(context, arguments)
        \texttt{shortdoc}
        \texttt{type} = \texttt{\textquote{user}}

class robot.running.userkeyword.\texttt{EmbeddedArgs}(name, template)
    Bases: robot.running.userkeyword.UserKeywordHandler
        \texttt{init\_keyword}(varz)
        \texttt{longname}
        \texttt{run}(context, arguments)
        \texttt{shortdoc}
        \texttt{type} = \texttt{\textquote{user}}

Subpackages

timeouts Package
timeouts Package
class robot.running.timeouts.TestTimeout (timeout=None, message=' ', variables=None)
    Bases: robot.running.timeouts._Timeout

    type = 'Test'

    set_keyword_timeout (timeout_occurred)

    any_timeout_occurred ()

    active

    get_message ()

    replace_variables (variables)

    run (runnable, args=None, kwargs=None)

    start ()

    time_left ()

    timed_out ()

class robot.running.timeouts.KeywordTimeout (timeout=None, message=' ', variables=None)
    Bases: robot.running.timeouts._Timeout

    type = 'Keyword'

    active

    get_message ()

    replace_variables (variables)

    run (runnable, args=None, kwargs=None)

    start ()

    time_left ()

    timed_out ()

stoppablethread Module
class robot.running.timeouts.stoppablethread.Thread (runner, name=None)
    Bases: threading.Thread

    A subclass of threading.Thread, with a stop() method.

    Original version posted by Connelly Barnes to python-list and available at http://mail.python.org/pipermail/python-list/2004-May/219465.html

    This version mainly has kill() changed to stop() to match java.lang.Thread.

    This is a hack but seems to be the best way the get this done. Only used in Python because in Jython we can use java.lang.Thread.

    start ()

    stop ()

    daemon

    A boolean value indicating whether this thread is a daemon thread (True) or not (False).

    This must be set before start() is called, otherwise Runtime Error is raised. Its initial value is inherited from the creating thread; the main thread is not a daemon thread and therefore all threads created in the main thread default to daemon = False.
The entire Python program exits when no alive non-daemon threads are left.

**getName()**

Thread identifier of this thread or None if it has not been started.

This is a nonzero integer. See the thread.get_ident() function. Thread identifiers may be recycled when a thread exits and another thread is created. The identifier is available even after the thread has exited.

**isAlive()**

Return whether the thread is alive.

This method returns True just before the run() method starts until just after the run() method terminates. The module function enumerate() returns a list of all alive threads.

**isDaemon()**

**is_alive()**

Return whether the thread is alive.

This method returns True just before the run() method starts until just after the run() method terminates. The module function enumerate() returns a list of all alive threads.

**join**(timeout=None)

Wait until the thread terminates.

This blocks the calling thread until the thread whose join() method is called terminates – either normally or through an unhandled exception or until the optional timeout occurs.

When the timeout argument is present and not None, it should be a floating point number specifying a timeout for the operation in seconds (or fractions thereof). As join() always returns None, you must call isAlive() after join() to decide whether a timeout happened – if the thread is still alive, the join() call timed out.

When the timeout argument is not present or None, the operation will block until the thread terminates. A thread can be join()ed many times.

join() raises a RuntimeError if an attempt is made to join the current thread as that would cause a deadlock. It is also an error to join() a thread before it has been started and attempts to do so raises the same exception.

**name**

A string used for identification purposes only.

It has no semantics. Multiple threads may be given the same name. The initial name is set by the constructor.

**run()**

Method representing the thread’s activity.

You may override this method in a subclass. The standard run() method invokes the callable object passed to the object’s constructor as the target argument, if any, with sequential and keyword arguments taken from the args and kwargs arguments, respectively.

**setDaemon**(daemonic)

**setName**(name)

### timeoutsignaling Module

`class robot.running.timeouts.timeoutsignaling.Timeout(timeout, error)`

Bases: object

**execute**(runnable)
timeoutthread Module

class robot.running.timeouts.timeoutthread.ThreadedRunner(runnable)
    Bases: object

    run()
    run_in_thread(timeout)
    get_result()
    stop_thread()

class robot.running.timeouts.timeoutthread.Timeout(timeout, error)
    Bases: object
    execute(runnable)

timeoutwin Module

class robot.running.timeouts.timeoutwin.Timeout(timeout, timeout_error)
    Bases: object
    execute(runnable)

utils Package

utils Package

Various generic utility classes and functions.
Provided utilities are generally stable, but absolute backwards compatibility between major versions is not guaranteed.

application Module

class robot.utils.application.Application(usage, name=None, version=None, arg_limits=None, logger=None, **auto_options)
    Bases: object

    main(arguments, **options)
    validate(options, arguments)
    execute_cli(cli_arguments)
    console(msg)
    execute(*arguments, **options)

class robot.utils.application.DefaultLogger
    Bases: object

    register_file_logger()
    info(message)
    error(message)
    close()
argumentparser Module

class robot.utils.argumentparser.ArgumentParser(usage, name=None, version=None, arg_limits=None, validator=None, auto_help=True, auto_version=True, auto_escape=True, auto_pythonpath=True, auto_argumentfile=True)

Available options and tool name are read from the usage.

Tool name is got from the first row of the usage. It is either the whole row or anything before first ‘–’.

parse_args(args_list)
Parse given arguments and return options and positional arguments.

Arguments must be given as a list and are typically sys.argv[1:]

Options are retuned as a dictionary where long options are keys. Value is a string for those options that can be given only one time (if they are given multiple times the last value is used) or None if the option is not used at all. Value for options that can be given multiple times (denoted with ‘*’ in the usage) is a list which contains all the given values and is empty if options are not used. Options not taken arguments have value False when they are not set and True otherwise.

Positional arguments are returned as a list in the order they are given.

If ‘check_args’ is True, this method will automatically check that correct number of arguments, as parsed from the usage line, are given. If the last argument in the usage line ends with the character ‘s’, the maximum number of arguments is infinite.

Possible errors in processing arguments are reported using DataError.

Some options have a special meaning and are handled automatically if defined in the usage and given from the command line:

–escape option can be used to automatically unescape problematic characters given in an escaped format.

–argumentfile can be used to automatically read arguments from a specified file. When –argumentfile is used, the parser always allows using it multiple times. Adding ‘*’ to denote that is thus recommend. A special value ‘stdin’ can be used to read arguments from stdin instead of a file.

–pythonpath can be used to add extra path(s) to sys.path.

–help and –version automatically generate help and version messages. Version is generated based on the tool name and version – see __init__ for information how to set them. Help contains the whole usage given to __init__. Possible <VERSION> text in the usage is replaced with the given version. Possible <-ESCAPES-> is replaced with available escapes so that they are wrapped to multiple lines but take the same amount of horizontal space as <-—ESCAPES—>. Both help and version are wrapped to Information exception.

class robot.utils.argumentparser.ArgLimitValidator(arg_limits)
Bases: object

class robot.utils.argumentparser.ArgFileParser(options)
Bases: object

process(args)

asserts Module

Convenience functions for testing both in unit and higher levels.
Benefits:

- Integrates 100% with unittest (see example below)
- Can be easily used without unittest (using unittest.TestCase when you only need convenient asserts is not so nice)
- Saved typing and shorter lines because no need to have ‘self.’ before asserts. These are static functions after all so that is OK.
- All ‘equals’ methods (by default) report given values even if optional message given. This behavior can be controlled with the optional values argument.

Drawbacks:

- unittest is not able to filter as much non-interesting traceback away as with its own methods because ASSERTionErrors occur outside

Most of the functions are copied more or less directly from unittest.TestCase which comes with the following license. Further information about unittest in general can be found from http://pyunit.sourceforge.net/. This module can be used freely in same terms as unittest.

unittest license:

Copyright (c) 1999-2003 Steve Purcell
This module is free software, and you may redistribute it and/or modify it under the same terms as Python itself, so long as this copyright message and disclaimer are retained in their original form.

IN NO EVENT SHALL THE AUTHOR BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF THIS CODE, EVEN IF THE AUTHOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

THE AUTHOR SPECIFICALLY DISCLAIMS ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE CODE PROVIDED HEREUNDER IS ON AN "AS IS" BASIS, AND THERE IS NO OBLIGATION WHATSOEVER TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

Examples:

```python
import unittest
from robot.util.asserts import *

class MyTests(unittest.TestCase):
    def test_old_style(self):
        self.assertEquals(1, 2, 'my msg')

    def test_new_style(self):
        assert_equals(1, 2, 'my msg')
```

Example output:

```
FF
======================================================================
FAIL: test_old_style (__main__.MyTests)
----------------------------------------------------------------------
Traceback (most recent call last):
File "example.py", line 7, in test_old_style
    self.assertEquals(1, 2, 'my msg')
```

4.1. robot Package
AssertionError: my msg

======================================================================
FAIL: test_new_style (__main__.MyTests)
======================================================================
Traceback (most recent call last):
File "example.py", line 10, in test_new_style
   assert_equals(1, 2, 'my msg')
File "/path/to/robot/asserts.py", line 142, in fail_unless_equal
   _report_unequality_failure(first, second, msg, values, '!=')
File "/path/to/robot/src/robot/asserts.py", line 209, in _report_unequality_failure
   raise _report_failure(msg)
File "/path/to/robot/src/robot/asserts.py", line 200, in _report_failure
   raise AssertionError(msg)
AssertionError: my msg: 1 != 2

Ran 2 tests in 0.000s
FAILED (failures=2)

robot.utils.asserts.fail(msg=None)
   Fail test immediately with the given message.

robot.utils.asserts.error(msg=None)
   Error test immediately with the given message.

robot.utils.asserts.fail_if(expr, msg=None)
   Fail the test if the expression is True.

robot.utils.asserts.fail_unless(expr, msg=None)
   Fail the test unless the expression is True.

robot.utils.asserts.fail_if_none(obj, msg=None, values=True)
   Fail the test if given object is None.

robot.utils.asserts.fail_unless_none(obj, msg=None, values=True)
   Fail the test if given object is not None.

robot.utils.asserts.fail_unless_raises(exc_class, callable_obj, *args, **kwargs)
   Fail unless an exception of class exc_class is thrown by callable_obj.

   callable_obj is invoked with arguments args and keyword arguments kwargs. If a different type of exception
   is thrown, it will not be caught, and the test case will be deemed to have suffered an error, exactly as for an
   unexpected exception.

   If a correct exception is raised, the exception instance is returned by this method.

robot.utils.asserts.fail_unless_raises_with_msg(exc_class, expected_msg, callable_obj, *args, **kwargs)
   Similar to fail_unless_raises but also checks the exception message.

robot.utils.asserts.fail_unless_equal(first, second, msg=None, values=True)
   Fail if given objects are unequal as determined by the ‘==’ operator.

robot.utils.asserts.fail_if_equal(first, second, msg=None, values=True)
   Fail if given objects are equal as determined by the ‘==’ operator.

robot.utils.asserts.fail_unless_almost_equal(first, second, places=7, msg=None, values=True)
   Fail if the two objects are unequal after rounded to given places.
Unequality is determined by object’s difference rounded to the given number of decimal places (default 7) and comparing to zero. Note that decimal places (from zero) are usually not the same as significant digits (measured from the most significant digit).

```
robot.utils.asserts.fail_if_almost_equal(first, second, places=7, msg=None, values=True)
```

Fail if the two objects are unequal after rounded to given places.

Equality is determined by object’s difference rounded to to the given number of decimal places (default 7) and comparing to zero. Note that decimal places (from zero) are usually not the same as significant digits (measured from the most significant digit).

```
robot.utils.asserts.assert_equal(first, second, msg=None, values=True)
```

Fail if given objects are unequal as determined by the ‘==’ operator.

```
robot.utils.asserts.assert_equals(first, second, msg=None, values=True)
```

Fail if given objects are unequal as determined by the ‘==’ operator.

```
robot.utils.asserts.assert_not_equal(first, second, msg=None, values=True)
```

Fail if given objects are equal as determined by the ‘==’ operator.

```
robot.utils.asserts.assert_not_equals(first, second, msg=None, values=True)
```

Fail if given objects are equal as determined by the ‘==’ operator.

```
robot.utils.asserts.assert_almost_equal(first, second, places=7, msg=None, values=True)
```

Fail if the two objects are unequal after rounded to given places.

Unequality is determined by object’s difference rounded to the given number of decimal places (default 7) and comparing to zero. Note that decimal places (from zero) are usually not the same as significant digits (measured from the most significant digit).

```
robot.utils.asserts.assert_almost_equals(first, second, places=7, msg=None, values=True)
```

Fail if the two objects are unequal after rounded to given places.

Unequality is determined by object’s difference rounded to the given number of decimal places (default 7) and comparing to zero. Note that decimal places (from zero) are usually not the same as significant digits (measured from the most significant digit).

```
robot.utils.asserts.assert_not_almost_equal(first, second, places=7, msg=None, values=True)
```

Fail if the two objects are unequal after rounded to given places.

Equality is determined by object’s difference rounded to to the given number of decimal places (default 7) and comparing to zero. Note that decimal places (from zero) are usually not the same as significant digits (measured from the most significant digit).

```
robot.utils.asserts.assert_not_almost_equals(first, second, places=7, msg=None, values=True)
```

Fail if the two objects are unequal after rounded to given places.

Equality is determined by object’s difference rounded to to the given number of decimal places (default 7) and comparing to zero. Note that decimal places (from zero) are usually not the same as significant digits (measured from the most significant digit).

```
robot.utils.asserts.assert_raises(exc_class, callable_obj, *args, **kwargs)
```

Fail unless an exception of class exc_class is thrown by callable_obj.

callable_obj is invoked with arguments args and keyword arguments kwargs. If a different type of exception is thrown, it will not be caught, and the test case will be deemed to have suffered an error, exactly as for an unexpected exception.

If a correct exception is raised, the exception instance is returned by this method.
robot.utils.asserts.assert_raises_with_msg(exc_class, expected_msg, callable_obj, *args, **kwargs)

Similar to fail_unless_raises but also checks the exception message.

robot.utils.asserts.assert_(expr, msg=None)

Fail the test unless the expression is True.

robot.utils.asserts.assert_true(expr, msg=None)

Fail the test unless the expression is True.

robot.utils.asserts.assert_false(expr, msg=None)

Fail the test if the expression is True.

robot.utils.asserts.assert_none(obj, msg=None, values=True)

Fail the test if given object is not None.

robot.utils.asserts.assert_not_none(obj, msg=None, values=True)

Fail the test if given object is None.

charwidth Module

A module to handle different character widths on the console.
Some East Asian characters have width of two on console, and combining characters themselves take no extra space.
See issue 604 [1] for more details about East Asian characters. The issue also contains generate_wild_chars.py script that was originally used to create _EAST_ASIAN_WILD_CHARS mapping. An updated version of the script is attached to issue 1096. Big thanks for xieyanbo for the script and the original patch.
Note that Python’s unicodedata module is not used here because importing it takes several seconds on Jython.


robot.utils.charwidth.get_char_width(char)

compress Module

robot.utils.compress.compress_text(text)

connectioncache Module

class robot.utils.connectioncache.ConnectionCache(no_current_msg='No open connection')

Connection cache for different Robot test libraries that use connections.
This cache stores connections and allows switching between them using generated indexes or user given aliases.
Can be used for example by web testing libraries where there’s need for multiple concurrent connections.
Note that in most cases there should be only one instance of this class but this is not enforced.

register(connection, alias=None)

Registers given connection with optional alias and returns its index.
Given connection is set to be the current connection. Alias must be a string. The index of the first connection after initialization or close_all or empty_cache is 1, second is 2, etc.
switch(index_or_alias)
Switches to the connection specified by given index or alias.
If alias is given it must be a string. Indexes can be either integers or strings that can be converted into integer. Raises RuntimeError if no connection with given index or alias found.

close_all(closer_method='close')
Closes connections using given closer method and empties cache.
If simply calling the closer method is not adequate for closing connections, clients should close connections themselves and use empty_cache afterwards.
empty_cache()
Empties the connections cache.
Indexes of new connections starts from 1 after this.

encoding Module

robot.utils.encoding.decode_output(string)
Decodes bytes from console encoding to Unicode.
robot.utils.encoding.encode_output(string, errors='replace')
Encodes Unicode to bytes in console encoding.
robot.utils.encoding.decode_from_system(string, can_be_from_java=True)
Decodes bytes from system (e.g. cli args or env vars) to Unicode.
robot.utils.encoding.encode_to_system(string, errors='replace')
Encodes Unicode to system encoding (e.g. cli args and env vars).
robot.utils.encoding.UTF8open(*args, **kwds)

encodingsniffer Module

robot.utils.encodingsniffer.get_system_encoding()
robot.utils.encodingsniffer.get_output_encoding()

error Module

robot.utils.error.get_error_message()
Returns error message of the last occurred exception.
This method handles also exceptions containing unicode messages. Thus it MUST be used to get messages from all exceptions originating outside the framework.
robot.utils.error.get_error_details()
Returns error message and details of the last occurred exception.
robot.utils.error.ErrorDetails()
This factory returns an object that wraps the last occurred exception
It has attributes message, traceback and error, where message contains type and message of the original error, traceback contains the traceback/stack trace and error contains the original error instance.

class robot.utils.error.PythonErrorDetails(exc_type, exc_value, exc_traceback)
Bases: robot.utils.error._ErrorDetails
message
traceback
class robot.utils.error.JavaErrorDetails(\texttt{exc\_type}, \texttt{exc\_value}, \texttt{exc\_traceback})
    Bases: robot.utils.error._ErrorDetails
    message
    traceback

\texttt{escaping} Module

robot.utils.escaping.\texttt{escape} \texttt{(item)}
robot.utils.escaping.\texttt{unescape} \texttt{(item)}

\texttt{etreewrapper} Module

class robot.utils.etreewrapper.\texttt{ETSource}(\texttt{source})
    Bases: object

\texttt{htmlformatters} Module

class robot.utils.htmlformatters.\texttt{LinkFormatter}
    Bases: object
    format_url \texttt{(text)}
    format_link \texttt{(text)}

class robot.utils.htmlformatters.\texttt{LineFormatter}
    Bases: object
    handles \texttt{(line)}
    newline = \texttt{\textasciitilde\textbackslash n}
    format \texttt{(line)}

class robot.utils.htmlformatters.\texttt{HtmlFormatter}
    Bases: object
    format \texttt{(text)}

class robot.utils.htmlformatters.\texttt{RulerFormatter}
    Bases: robot.utils.htmlformatters._SingleLineFormatter
    format_line \texttt{(line)}
    add \texttt{(line)}
    end()
    format \texttt{(lines)}
    handles \texttt{(line)}

class robot.utils.htmlformatters.\texttt{HeaderFormatter}
    Bases: robot.utils.htmlformatters._SingleLineFormatter
    format_line \texttt{(line)}
Robot Framework Documentation, Release 2.7.7

add(line)
end()
format(lines)
handles(line)

class robot.utils.htmlformatters.ParagraphFormatter(other_formatters)
    Bases: robot.utils.htmlformatters._Formatter
    format(lines)
    add(line)
    end()
    handles(line)

class robot.utils.htmlformatters.TableFormatter
    Bases: robot.utils.htmlformatters._Formatter
    format(lines)
    add(line)
    end()
    handles(line)

class robot.utils.htmlformatters.PreformattedFormatter
    Bases: robot.utils.htmlformatters._Formatter
    format(lines)
    add(line)
    end()
    handles(line)

class robot.utils.htmlformatters.ListFormatter
    Bases: robot.utils.htmlformatters._Formatter
    format(lines)
    add(line)
    end()
    handles(line)

importer Module

class robot.utils.importer.Importer(type=None, logger=None)
    Bases: object
    import_class_or_module(name, instantiate_with_args=None)
        Imports Python class/module or Java class with given name.
        Class can either live in a module/package or be standalone Java class. In the former case the name
        is something like ‘MyClass’ and in the latter it could be ‘your.package.YourLibrary’. Python classes always
        live in a module, but if the module name is exactly same as the class name then simple ‘MyLibrary’ will
        import a class.
        Python modules can be imported both using format ‘MyModule’ and ‘mymodule.submodule’.

4.1. robot Package
name can also be a path to the imported file/directory. In that case importing is done using import_class_or_module_by_path method.

If instantiate_with_args is not None, imported classes are instantiated with the specified arguments automatically.

import_class_or_module_by_path (path, instantiate_with_args= None)
Import a Python module or Java class using a file system path.

When importing a Python file, the path must end with `.py` and the actual file must also exist. When importing a Python module implemented as a directory, the path must end with `/' or, on Windows, with `\`.

When importing Java classes, the path must end with `.java` or `.class`. The class file must exist in both cases and in the former case also the source file must exist.

If instantiate_with_args is not None, imported classes are instantiated with the specified arguments automatically.

class robot.utils.importer.ByPathImporter (logger)
Bases: robot.utils.importer._Importer

handles (path)

import_ (path)

class robot.utils.importer.NonDottedImporter (logger)
Bases: robot.utils.importer._Importer

handles (name)

import_ (name)

class robot.utils.importer.DottedImporter (logger)
Bases: robot.utils.importer._Importer

handles (name)

import_ (name)

markuputils Module

robot.utils.markuputils.html_escape (text)
robot.utils.markuputils.xml_escape (text)
robot.utils.markuputils.html_format (text)
robot.utils.markuputils.attribute_escape (attr)

markupwriters Module

class robot.utils.markupwriters.HtmlWriter (output, line_separator= 'n', encoding= None)
Bases: robot.utils.markupwriters._MarkupWriter

Parameters

• output – Either an opened, file like object, or a path to the desired output file. In the latter case, the file is created and clients should use close() method to close it.

• line_separator – Defines the used line separator.
• **encoding** – Encoding to be used to encode all text written to the output file. If `None`, text will not be encoded.

```python
close()
    Closes the underlying output file.
content(content=None, escape=True)
    element(name, content=None, attrs=None, escape=True, newline=True)
    end(name, newline=True)
    start(name, attrs=None, newline=True)
```

**class** robot.utils.markupwriters.XmlWriter(output, line_separator='n', encoding=None)
Bases: robot.utils.markupwriters._MarkupWriter
    close()
    Closes the underlying output file.
    content(content=None, escape=True)
    element(name, content=None, attrs=None, escape=True, newline=True)
    end(name, newline=True)
    start(name, attrs=None, newline=True)

**class** robot.utils.markupwriters.NullMarkupWriter(*args)
Bases: object
    Null implementation of _MarkupWriter interface
    start(*args)
    content(*args)
    element(*args)
    end(*args)
    close(*args)
```

**match Module**

robot.utils.match.eq(str1, str2, ignore=(), caseless=True, spaceless=True)
robot.utils.match.matches(string, pattern, ignore=(), caseless=True, spaceless=True)
    Deprecated!! Use Matcher instead.
robot.utils.match.matches_any(string, patterns, ignore=(), caseless=True, spaceless=True)
    Deprecated!! Use MultiMatcher instead.

**class** robot.utils.match.Matcher(pattern, ignore=(), caseless=True, spaceless=True)
Bases: object
    match(string)

**class** robot.utils.match.MultiMatcher(patterns=None, ignore=(), caseless=True, spaceless=True, match_if_no_patterns=False)
Bases: object
    match(string)
misc Module

robot.utils.misc.printable_name(string, code_style=False)
Generates and returns printable name from the given string.
Examples:
‘simple’ -> ‘Simple’
‘name with spaces’ -> ‘Name With Spaces’
‘more spaces’ -> ‘More Spaces’
‘Cases AND spaces’ -> ‘Cases AND Spaces’
‘ straight’ -> ‘ Straight’
If ‘code_style’ is True:
‘mixedCAPSCamel’ -> ‘Mixed CAPS Camel’
‘camelCaseName’ -> ‘Camel Case Name’
‘under_score_name’ -> ‘Under_score_name’
‘under_and space’ -> ‘Under and space’
‘more spaces’ -> ‘More Spaces’
‘Cases AND spaces’ -> ‘Cases AND Spaces’
‘straight’ -> ‘ Straight’

robot.utils.misc.plural_or_not(item)

robot.utils.misc.seq2str(sequence, quote='', sep=' ', lastsep=' and ')
Returns sequence in format ‘item 1’, ‘item 2’ and ‘item 3’

robot.utils.misc.seq2str2(sequence)
Returns sequence in format [ item 1 | item 2 | ... ]

robot.utils.misc.getdoc(item)

robot.utils.misc.isatty(stream)

normalizing Module

robot.utils.normalizing.normalize(string, ignore=(), caseless=True, spaceless=True)
Normalizes given string according to given spec.
By default string is turned to lower case and all whitespace is removed. Additional characters can be removed
by giving them in ignore list.

robot.utils.normalizing.lower(string)

robot.utils.normalizing.normalize_tags(tags)
DEPRECATED!! Use robot.model.Tags instead.
Returns tags sorted and duplicates, empty, and NONE removed.
If duplicate tags have different case/space, the one used first wins.

class robot.utils.normalizing.NormalizedDict(initial=None, ignore=(), caseless=True, spaceless=True)

Bases: UserDict.UserDict
Custom dictionary implementation automatically normalizing keys.
Initializes with possible initial value and normalizing spec.
Initial values can be either a dictionary or an iterable of name/value pairs. In the latter case items are added in
the given order.
Normalizing spec has exact same semantics as with normalize method.

update (dict=None, **kwargs)
set (key, value)
get (key, default=None)
pop (key)
clear()
has_key(key)
keys()
iterkeys()
values()
itervalues()
items()
iteritems()
copy()
classmethod fromkeys(iterable, value=None)
popitem()
setdefault(key, failobj=None)

robotenv Module

robot.utils.robotenv.get_env_var(name, default=None)
robot.utils.robotenv.set_env_var(name, value)
robot.utils.robotenv.del_env_var(name)
robot.utils.robotenv.get_env_vars()

robotpath Module

robot.utils.robotpath.normpath(path)
Returns path in normalized and absolute format.
On case-insensitive file systems the path is also case normalized. If that is not desired, abspath should be used instead.

robot.utils.robotpath.abspath(path)
Replacement for os.path.abspath with some bug fixes and enhancements.
1. Converts non-Unicode paths to Unicode using file system encoding
2. At least Jython 2.5.1 on Windows returns wrong path with ‘c:’.
3. Python until 2.6.5 and at least Jython 2.5.1 don’t handle non-ASCII characters in the working directory:
   http://bugs.python.org/issue3426

robot.utils.robotpath.get_link_path(target, base)
Returns a relative path to a target from a base.
If base is an existing file, then its parent directory is considered. Otherwise, base is assumed to be a directory.
Rationale: os.path.relpath is not available before Python 2.6

4.1. robot Package
**robottime Module**

*robot.utils.robottime.timestr_to_secs(timestr)*

Parses time in format like ‘1h 10s’ and returns time in seconds (float).

Given time must be in format ‘1d 2h 3m 4s 5ms’ with following rules:

- Time parts having zero value can be ignored (e.g. ‘3m 4s’ is ok)
- Format is case and space insensitive
- Instead of ‘d’ it is also possible to use ‘day’ or ‘days’
- Instead of ‘h’ also ‘hour’ and ‘hours’ are ok
- Instead of ‘m’ also ‘minute’, ‘minutes’, ‘min’ and ‘mins’ are ok
- Instead of ‘s’ also ‘second’, ‘seconds’, ‘sec’ and ‘secs’ are ok
- Instead of ‘ms’ also ‘millisecond’, ‘milliseconds’ and ‘millis’ are ok
- It is possible to give time only as a float and then it is considered to be seconds (e.g. ‘123’, ‘123.0’, ‘123s’, ‘2min 3s’ are all equivalent)

*robot.utils.robottime.secs_to_timestr(secs, compact=False)*

Converts time in seconds to a string representation.

Returned string is in format like ‘1 day 2 hours 3 minutes 4 seconds 5 milliseconds’ with following rules:

- Time parts having zero value are not included (e.g. ‘3 minutes 4 seconds’ instead of ‘0 days 0 hours 3 minutes 4 seconds’)
- Hour part has a maximum of 23 and minutes and seconds both have 59 (e.g. ‘1 minute 40 seconds’ instead of ‘100 seconds’)

If compact has value ‘True’, short suffixes are used. (e.g. 1d 2h 3min 4s 5ms)

*robot.utils.robottime.format_time(timetuple_or_epochsecs, daysep='', daytimesep=' ', time_sep=':', millissep=None, gmtsep=None)*

Returns a timestamp formatted from given time using separators.

Time can be given either as a timetuple or seconds after epoch.

Timetuple is (year, month, day, hour, min, sec[, millis]), where parts must be integers and millis is required only when millissep is not None. Notice that this is not 100% compatible with standard Python timetuples which do not have millis.

Seconds after epoch can be either an integer or a float.

*robot.utils.robottime.get_time(format='timestamp', time_=None)*

Return the given or current time in requested format.

If time is not given, current time is used. How time is returned is determined based on the given ‘format’ string as follows. Note that all checks are case insensitive.

- If ‘format’ contains word ‘epoch’ the time is returned in seconds after the unix epoch.
- If ‘format’ contains any of the words ‘year’, ‘month’, ‘day’, ‘hour’, ‘min’ or ‘sec’ only selected parts are returned. The order of the returned parts is always the one in previous sentence and order of words in ‘format’ is not significant. Parts are returned as zero padded strings (e.g. May -> ‘05’).
- Otherwise (and by default) the time is returned as a timestamp string in format ‘2006-02-24 15:08:31’

*robot.utils.robottime.parse_time(timestr)*

Parses the time string and returns its value as seconds since epoch.
Time can be given in five different formats:

1. Numbers are interpreted as time since epoch directly. It is possible to use also ints and floats, not only strings containing numbers.
2. Valid timestamp (‘YYYY-MM-DD hh:mm:ss’ and ‘YYYYMMDD hhmmss’).
3. ‘NOW’ (case-insensitive) is the current local time.
4. ‘UTC’ (case-insensitive) is the current time in UTC.
5. Format ‘NOW - 1 day’ or ‘UTC + 1 hour 30 min’ is the current local/UTC time plus/minus the time specified with the time string.

Seconds are rounded down to avoid getting times in the future.

```python
robot.utils.robotime.get_timestamp(daysep='', daytimesep=' ', timesep=':', millissep='.')
robot.utils.robotime.timestamp_to_secs(timestamp, seps=None)
robot.utils.robotimesecs_to_timestamp(secs, seps=None, millis=False)
robot.utils.robotime.get_start_timestamp(daysep='', daytimesep=' ', timesep=':', millissep=None)
robot.utils.robotime.get_elapsed_time(start_time, end_time)
    Returns the time between given timestamps in milliseconds.
robot.utils.robotime elapsed_time_to_string(elapsed, include_millis=True)
    Converts elapsed time in milliseconds to format ‘hh:mm:ss.mil’.
        If include_millis is True, ‘.mil’ part is omitted.
class robot.utils.robotime.TimestampCache
    Bases: object
        get_timestamp(daysep='', daytimesep=' ', timesep=':', millissep='.')

setter Module

class robot.utils.setter.setter(method)
    Bases: object
class robot.utils.setter.SetterAwareType
    Bases: type
        mro() → list
            return a type’s method resolution order

text Module

robot.utils.text.cut_long_message(msg)
robot.utils.text.format_assign_message(variable, value, cut_long=True)
robot.utils.text.get_console_length(text)
robot.utils.text.pad_console_length(text, width)
```
unic Module

robot.utils.unic.unic(item, *args)
robot.utils.unic.safe_repr(item)

variables Package

variables Package

Implements handling and resolving of variables.
This package is likely to change radically in RF 2.8. External code should use functionality provided directly by this package with care.
robot.variables.init_global_variables(settings)

isvar Module

robot.variables.isvar.is_var(string)
robot.variables.isvar.is_scalar_var(string)
robot.variables.isvar.is_list_var(string)

variableassigner Module

class robot.variables.variableassigner.VariableAssigner(assign)
    Bases: object
    assign(context, return_value)

class robot.variables.variableassigner.AssignParser(assign)
    Bases: object

class robot.variables.variableassigner.ReturnValue(scalar_vars, list_var, return_value)
    Bases: object
    get_variables_to_set()

variables Module

robot.variables.variables.getJavaSystemProperty(name)

class robot.variables.variables.Map

class robot.variables.variables.Variables(identifiers=('$', '@', '%', '&', '*'))
    Bases: robot.utils.normalizing.NormalizedDict
    Represents a set of variables including both ${scalars} and @{lists}.
    Contains methods for replacing variables from list, scalars, and strings. On top of ${scalar} and @{list} variables these methods handle also %{environment} variables.
    update(dict=None, **kwargs)
replace_list (items)
   Replaces variables from a list of items.
   If an item in a list is a @list variable its value is returned. Possible variables from other items are
   replaced using ‘replace_scalar’. Result is always a list.

replace_scalar (item)
   Replaces variables from a scalar item.
   If the item is not a string it is returned as is. If it is a ${scalar} variable its value is returned. Otherwise
   variables are replaced with ‘replace_string’. Result may be any object.

replace_string (string, splitted=None, ignore_errors=False)
   Replaces variables from a string. Result is always a string.

set_from_file (path, args=None, overwrite=False)

set_from_variable_table (variable_table, overwrite=False)

has_key (variable)
contains (variable, extended=False)
clear ()
copy ()
classmethod fromkeys (iterable, value=None)

get (key, default=None)

items ()
iteritems ()
iterkeys ()
itervalues ()
keys ()
pop (key)
popitem ()
set (key, value)
setdefault (key, failobj=None)
values ()

variablesplitter Module

class robot.variables.variablesplitter.VariableSplitter (string, identifiers)

   get_replaced_base (variables)

writer Package

writer Package

Implements writing of parsed test data to files.

4.1. robot Package
This functionality is used by `robot.parsing.model.TestCaseFile.save()`.

This package is considered stable.

### aligners Module

```python
class robot.writer.aligners.FirstColumnAligner(first_column_width):
    Bases: robot.writer.aligners._Aligner
    align_row(row)
    align_rows(rows)

class robot.writer.aligners.ColumnAligner(first_column_width, table):
    Bases: robot.writer.aligners._Aligner
    align_row(row)
    align_rows(rows)

class robot.writer.aligners_NullAligner(widths=None):
    Bases: robot.writer.aligners._Aligner
    align_row(row)
    align_rows(rows)
```

### dataextractor Module

```python
class robot.writer.dataextractor.DataExtractor(want_name_on_first_row=None):
    Bases: object
    Transforms table of a parsed test data file into a list of rows.
    rows_from_table(table)
```

### datafilewriter Module

```python
class robot.writer.datafilewriter.DataFileWriter(**options):
    Bases: object
    Object to write parsed test data file objects back to disk.

    Parameters **options – A WritingContext is created based
    on these.

    write(datafile)
    Writes given datafile using **options.

    Parameters datafile (TestCaseFile, ResourceFile, TestDataDirectory) – The
    parsed test data object to be written

class robot.writer.datafilewriter.WritingContext(datafile, format='', output=None, pipe_separated=False,
                                               txt_separating_spaces=4, line_separator='n')
```

Contains configuration used in writing a test data file to disk.
Parameters

- **datafile** ([`TestCaseFile`, `ResourceFile`, `TestDataDirectory`]) – The datafile to be written.
- **format** (`str`) – Output file format. If omitted, read from the extension of the `source` attribute of the given `datafile`.
- **output** – An open, file-like object used in writing. If omitted, value of `source` attribute of the given `datafile` is used to construct a new file object.
- **pipe_separated** (`bool`) – Whether to use pipes as separator when output file format is `txt`.
- **txt_separating_spaces** (`int`) – Number of separating spaces between cells in space separated format.
- **line_separator** (`str`) – Line separator used in output files.

If `output` is not given, an output file is created based on the source of the given datafile and value of `format`. Examples:

Write output in a `StringIO` instance using format of `datafile.source`:

```python
WriteConfiguration(datafile, output=StringIO)
```

Output file is created from `datafile.source` by stripping extension and replacing it with `html`:

```python
WriteConfiguration(datafile, format='html')
```

```python
encoding = 'UTF-8'
txt_format = 'txt'
html_format = 'html'
tsv_format = 'tsv'
robot_format = 'robot'
txt_column_count = 8
html_column_count = 5
tsv_column_count = 8
```

**filewriters Module**

```python
robot.writer.filewriters.FileWriter(context)
Creates and returns a FileWriter object.

Parameters
context (WritingContext) – Type of returned FileWriter is determined based on context.format. context is also passed to created writer.
```

```python
class robot.writer.filewriters.SpaceSeparatedTxtWriter(configuration)
Bases: robot.writer.filewriters._DataFileWriter

write(datafile)
```

```python
class robot.writer.filewriters.PipeSeparatedTxtWriter(configuration)
Bases: robot.writer.filewriters._DataFileWriter

write(datafile)
```

```python
class robot.writer.filewriters.TsvFileWriter(configuration)
Bases: robot.writer.filewriters._DataFileWriter
```

4.1. `robot` Package
write (datafile)

class robot.writer.filewriters.HtmlFileWriter (configuration)
    Bases: robot.writer.filewriters._DataFileWriter
    write (datafile)

formatters Module

class robot.writer.formatters.TsvFormatter (column_count)
    Bases: robot.writer.formatters._DataFileFormatter
    empty_row_after (table)
    format_header (table)
    format_table (table)

class robot.writer.formatters.TxtFormatter (column_count)
    Bases: robot.writer.formatters._DataFileFormatter
    empty_row_after (table)
    format_header (table)
    format_table (table)

class robot.writer.formatters.PipeFormatter (column_count)
    Bases: robot.writer.formatters.TxtFormatter
    empty_row_after (table)
    format_header (table)
    format_table (table)

htmlformatter Module

class robot.writer.htmlformatter.HtmlFormatter (column_count)
    Bases: robot.writer.formatters._DataFileFormatter
    format_header (table)
    empty_row_after (table)
    format_table (table)

class robot.writer.htmlformatter.HtmlCell (content='', attributes=None, tag='td', escape=True)
    Bases: object

class robot.writer.htmlformatter.NameCell (name='', attributes=None)
    Bases: robot.writer.htmlformatter.HtmlCell

class robot.writer.htmlformatter.AnchorNameCell (name, type_)
    Bases: robot.writer.htmlformatter.HtmlCell

class robot.writer.htmlformatter.DocumentationCell (content, span)
    Bases: robot.writer.htmlformatter.HtmlCell

class robot.writer.htmlformatter.HeaderCell (name, span=1)
    Bases: robot.writer.htmlformatter.HtmlCell
htmltemplate Module

rowsplitter Module

class robot.writer.rowsplitter.RowSplitter(cols=8, split_multiline_doc=True)
    Bases: object
    split(row, table_type)

4.2 api Package

4.2.1 api Package

This package exposes the public APIs of Robot Framework.

Unless stated otherwise, the APIs exposed in this module are considered stable, and thus safe to use when building external tools on top of Robot Framework.

Currently exposed APIs are:

• logger for test libraries’ logging purposes.
• ExecutionResult() for reading execution results from XML output files.
• TestCaseFile, TestDataDirectory, and ResourceFile for parsing data files. In addition, a convenience factory function TestData() creates either TestCaseFile or TestDataDirectory based on the input.
• TestSuite() for creating a test suite that can be executed. This API is going to change in Robot Framework 2.8.

These names can be imported like this:

    from robot.api import <name>

See documentations of the individual APIs for more details.

4.2.2 logger Module

Public logging API for test libraries.

This module provides a public API for writing messages to the log file and the console. Test libraries can use this API like logger.info('My message') instead of logging through the standard output like print '{INFO} My message'. In addition to a programmatic interface being cleaner to use, this API has a benefit that the log messages have accurate timestamps.

Log levels

It is possible to log messages using levels TRACE, DEBUG, INFO and WARN either using the write method or, more commonly, with the log level specific trace, debug, info and warn methods.

By default the trace and debug messages are not logged but that can be changed with the –loglevel command line option. Warnings are automatically written also to the Test Execution Errors section in the log file and to the console.
Logging HTML

All methods that are used for writing messages to the log file have an optional `html` argument. If a message to be logged is supposed to be shown as HTML, this argument should be set to `True`.

Example

```python
from robot.api import logger

def my_keyword(arg):
    logger.debug('Got argument %s' % arg)
do_something()
logger.info('This is a boring example', html=True)
```

```python
robot.api.logger.write(msg, level, html=False)
```

Writes the message to the log file using the given level.

Valid log levels are `TRACE`, `DEBUG`, `INFO` and `WARN`. Instead of using this method, it is generally better to use the level specific methods such as `info` and `debug`.

```python
robot.api.logger.trace(msg, html=False)
```

Writes the message to the log file with the TRACE level.

```python
robot.api.logger.debug(msg, html=False)
```

Writes the message to the log file with the DEBUG level.

```python
robot.api.logger.info(msg, html=False, also_console=False)
```

Writes the message to the log file with the INFO level.

If `also_console` argument is set to `True`, the message is written both to the log file and to the console.

```python
robot.api.logger.warn(msg, html=False)
```

Writes the message to the log file with the WARN level.

```python
robot.api.logger.console(msg, newline=True)
```

Writes the message to the console.

If the `newline` argument is `True`, a newline character is automatically added to the message.

4.3 common Package

4.3.1 common Package

Base classes for test execution model.

This code was earlier used also by test result processing modules but not anymore in RF 2.7.

The whole package is likely to be removed in RF 2.8 when test execution model is refactored. No new code should depend on this package.

4.3.2 handlers Module

```python
class robot.common.handlers.UserErrorHandler(name, error)
```

Created if creating handlers fail – running raises DataError.

The idea is not to raise DataError at processing time and prevent all tests in affected test case file from executing. Instead UserErrorHandler is created and if it is ever run DataError is raised then.

```python
type = 'error'
```
4.3.3 keyword Module

```python
class BaseKeyword(name='', args=None, doc='', timeout='', type='kw')

   passed
   serialize(serializer)
```

4.3.4 libraries Module

```python
class BaseLibrary

   get_handler(name)
   has_handler(name)
```

4.3.5 model Module

```python
class BaseTestSuite(name, source=None, parent=None)

   Bases: robot.common.model._TestAndSuiteHelper

   Base class for TestSuite used in runtime and but not anymore by rebot.

   set_name(name)

   name
   id

   set_critical_tags(critical, non_critical)

   set_doc(doc)

   set_metadata(metalist)

   get_test_count()

   get_full_message()
       Returns suite’s message including statistics message

   get_stat_message()

   set_status()
       Sets status and statistics based on subsuite and test statuses.

       Can/should be used when statuses have been changed somehow.

   suite_teardown_failed(error=None, message=None)

   set_tags(tags)

   filter(suites=None, tests=None, includes=None, excludes=None, zero_tests_ok=False)

   filter_by_names(suites=None, tests=None, zero_tests_ok=False)

   filter_by_tags(includes=None, excludes=None, zero_tests_ok=False)
```

4.3. common Package
set_runmode(runmode)
set_options(settings)
serialize(serializer)
return_code

longname

class robot.common.model.BaseTestCase(name, parent)
Bases: robot.common.model._TestAndSuiteHelper

id

suite_teardown_failed(message)

set_criticality(critical)

is_included(incl_tags, excl_tags)
    Returns True if this test case is included but not excluded.
    If no 'incl_tags' are given all tests are considered to be included.

serialize(serializer)

longname

4.3.6 statistics Module

class robot.common.statistics.Statistics(suite, suite_stat_level=-1, tag_stat_include=None,
tag_stat_exclude=None, tag_stat_combine=None, tag_doc=None, tag_stat_link=None)

serialize(serializer)

class robot.common.statistics.Stat(name='')

total

add_stat(other)

add_test(test)

fail_all()

add_suite(suite)

class robot.common.statistics.CriticalStats(suite)
Bases: robot.common.statistics.Stat

add_stat(other)

add_suite(suite)

add_test(test)

fail_all()

total

class robot.common.statistics>AllStats(suite)
Bases: robot.common.statistics.Stat
add_stat (other)
add_suite (suite)
add_test (test)
fail_all()
total
class robot.common.statistics.SuiteStat (suite)
    Bases: robot.common.statistics.Stat
    type = 'suite'
    serialize (serializer)
    add_stat (other)
    add_suite (suite)
    add_test (test)
    fail_all()
    total
class robot.common.statistics.TagStat (name, doc='', links=[], critical=False, non_critical=False, combined='')
    Bases: robot.common.statistics.Stat
    type = 'tag'
    add_test (test)
    serialize (serializer)
    add_stat (other)
    add_suite (suite)
    fail_all()
    total
class robot.common.statistics.TotalStat (name, suite_stat)
    Bases: robot.common.statistics.Stat
    type = 'total'
    serialize (serializer)
    add_stat (other)
    add_suite (suite)
    add_test (test)
    fail_all()
    total
class robot.common.statistics.SuiteStatistics (suite, tag_stats, suite_stat_level=-1)
        serialize (serializer)
class robot.common.statistics.TagStatistics (include=None, exclude=None, combine=None, docs=None, links=None)
add_test (test, critical)
serialize (serializer)
sort ()
class robot.common.statistics.TotalStatistics (suite)

serialize (serializer)
class robot.common.statistics.TagStatInfo (docs, links)

get_doc (tag)
get_links (tag)
class robot.common.statistics.TagStatDoc (pattern, doc)

matches (tag)
class robot.common.statistics.TagStatLink (pattern, link, title)

matches (tag)
get_link (tag)

4.4 conf Package

4.4.1 conf Package

Settings for both test execution and output processing.

4.4.2 settings Module

class robot.conf.settings.RobotSettings (options=None, log=True)
Bases: robot.conf.settings._BaseSettings

is_rebot_needed ()
get_rebot_datasource_and_settings ()
log
output
report
split_log
status_rc
xunit
class robot.conf.settings.RebotSettings (options=None, log=True)
Bases: robot.conf.settings._BaseSettings

suite_config
statistics_config
4.5 htmldata Package

4.5.1 htmldata Package

Generic functionality for writing output files in HTML format.

robot.reporting, robot.libdoc and robot.testdoc use this package.

4.5.2 htmlfilewriter Module

```python
class robot.htmldata.htmlfilewriter.HtmlFileWriter(output, model_writer)
    Bases: object
    write(template)

class robot.htmldata.htmlfilewriter.ModelWriter
    Bases: robot.htmldata.htmlfilewriter._Writer
    handles(line)
    write(line)

class robot.htmldata.htmlfilewriter.LineWriter(output)
    Bases: robot.htmldata.htmlfilewriter._Writer
    handles(line)
    write(line)

class robot.htmldata.htmlfilewriter.GeneratorWriter(html_writer)
    Bases: robot.htmldata.htmlfilewriter._Writer
    write(line)
    handles(line)

class robot.htmldata.htmlfilewriter.JsFileWriter(html_writer, base_dir)
    Bases: robot.htmldata.htmlfilewriter._InliningWriter
    write(line)
    handles(line)

class robot.htmldata.htmlfilewriter.CssFileWriter(html_writer, base_dir)
    Bases: robot.htmldata.htmlfilewriter._InliningWriter
    write(line)
```
handles (line)

4.5.3 jartemplate Module

4.5.4 jsonwriter Module

class robot.htmldata.jsonwriter.JsonWriter (output, separator='')
  Bases: object
  write_json (prefix, data, postfix=';n', mapping=None, separator=True)
  write (string, postfix=';n', separator=True)

class robot.htmldata.jsonwriter.JsonDumper (output)
  Bases: object
  dump (data, mapping=None)
  write (data)

class robot.htmldata.jsonwriter.StringDumper (jsondumper)
  Bases: robot.htmldata.jsonwriter._Dumper
  dump (data, mapping)
  handles (data, mapping)

class robot.htmldata.jsonwriter.IntegerDumper (jsondumper)
  Bases: robot.htmldata.jsonwriter._Dumper
  dump (data, mapping)
  handles (data, mapping)

class robot.htmldata.jsonwriter.DictDumper (jsondumper)
  Bases: robot.htmldata.jsonwriter._Dumper
  dump (data, mapping)
  handles (data, mapping)

class robot.htmldata.jsonwriter.TupleListDumper (jsondumper)
  Bases: robot.htmldata.jsonwriter._Dumper
  dump (data, mapping)
  handles (data, mapping)

class robot.htmldata.jsonwriter.MappingDumper (jsondumper)
  Bases: robot.htmldata.jsonwriter._Dumper
  handles (data, mapping)
  dump (data, mapping)

class robot.htmldata.jsonwriter.NoneDumper (jsondumper)
  Bases: robot.htmldata.jsonwriter._Dumper
  handles (data, mapping)
  dump (data, mapping)
4.5.5 normaltemplate Module

class robot.htmldata.normaltemplate.HtmlTemplate(filename)
    Bases: object

4.5.6 template Module

4.6 libdocpkg Package

4.6.1 libdocpkg Package

Implements libdoc tool.
For programmatic entry point, see robot.libdoc.
This package is considered stable.

robot.libdocpkg.LibraryDocumentation(library_or_resource, name=None, version=None, doc_format=None)

4.6.2 builder Module

robot.libdocpkg.builder.JavaDocBuilder()
robot.libdocpkg.builder.DocumentationBuilder(library_or_resource)

4.6.3 consoleviewer Module

class robot.libdocpkg.consoleviewer.ConsoleViewer(libdoc)
    Bases: object
        classmethod handles(command)
        classmethod validate_command(command, args)
        view(command, *args)
        list(*patterns)
        show(*names)
        version()

class robot.libdocpkg.consoleviewer.KeywordMatcher(libdoc)
    Bases: object
        search(*patterns)

4.6.4 htmlwriter Module

class robot.libdocpkg.htmlwriter.LibdocHtmlWriter
    Bases: object
        write(libdoc, output)
class robot.libdocpkg.htmlwriter.LibdocModelWriter(output, libdoc)
    Bases: robot.htmldata.htmlfilewriter.ModelWriter
    write(line)
    write_data()
    handles(line)

class robot.libdocpkg.htmlwriter.JsonConverter(doc_formatter)
    Bases: object
    convert(libdoc)

class robot.libdocpkg.htmlwriter.DocFormatter(keywords, introduction, doc_format='ROBOT')
    Bases: object
    html(doc, intro=False)

class robot.libdocpkg.htmlwriter.DocToHtml(format)
    Bases: object

4.6.5 javabuilder Module

class robot.libdocpkg.javabuilder.JavaDocBuilder
    Bases: object
    build(path)

robot.libdocpkg.javabuilder.ClassDoc(path)
    Process the given Java source file and return ClassDoc instance.
    Processing is done using com.sun.tools.javadoc APIs. The usage has been figured out from sources at
    Returned object implements com.sun.javadoc.ClassDoc interface, see http://java.sun.com/j2se/1.4.2/docs/tooldocs/javadoc/doclet/

4.6.6 model Module

class robot.libdocpkg.model.LibraryDoc(name='', doc='', version='', type='library', scope='', named_args=False, doc_format='')
    Bases: object
    scope
doc_format
keywords
save(output=None, format='HTML')

class robot.libdocpkg.model.KeywordDoc(name='', args=None, doc='')
    Bases: object
    shortdoc

4.6.7 output Module

class robot.libdocpkg.output.LibdocOutput(output_path, format)
    Bases: object
4.6.8 robotbuilder Module

```python
class robot.libdocpkg.robotbuilder.LibraryDocBuilder
    Bases: object
    build(library)

class robot.libdocpkg.robotbuilder.ResourceDocBuilder
    Bases: object
    build(path)

class robot.libdocpkg.robotbuilder.KeywordDocBuilder
    Bases: object
    build_keywords(lib)
    build_keyword(kw)
```

4.6.9 specbuilder Module

```python
class robot.libdocpkg.specbuilder.SpecDocBuilder
    Bases: object
    build(path)
```

4.6.10 writer Module

```python
robot.libdocpkg.writer.LibdocWriter(format=None)
```

4.6.11 xmlwriter Module

```python
class robot.libdocpkg.xmlwriter.LibdocXmlWriter
    Bases: object
    write(libdoc, outfile)
```

4.7 libraries Package

4.7.1 libraries Package

Contains Robot Framework standard test libraries.

4.7.2 BuiltIn Module

```python
class robot.libraries.BuiltIn
    Bases: robot.libraries.BuiltIn._Verify, robot.libraries.BuiltIn._Converter,
            robot.libraries.BuiltIn._Variables, robot.libraries.BuiltIn._RunKeyword,
            robot.libraries.BuiltIn._Misc
    An always available standard library with often needed keywords.
```
BuiltIn is Robot Framework’s standard library that provides a set of generic keywords needed often. It is imported automatically and thus always available. The provided keywords can be used, for example, for verifications (e.g. Should Be Equal, Should Contain), conversions (e.g. Convert To Integer) and for various other purposes (e.g. Log, Sleep, Run Keyword If, Set Global Variable).

**ROBOT_LIBRARY_SCOPE** = ‘GLOBAL’

**ROBOT_LIBRARY_VERSION** = ‘2.7.7’

call_method (object, method_name, *args)

Calls the named method of the given object with the provided arguments.

The possible return value from the method is returned and can be assigned to a variable. Keyword fails both if the object does not have a method with the given name or if executing the method raises an exception.

Examples:

| Call Method | ${hashtable} | put | myname | myvalue | | | ${isempty} = | Call Method | ${hashtable} | isEmpty | | | | | Should Not Be True | ${isempty} | | | | | | | | | ${value} = | Call Method | ${hashtable} | get | myname | | Should Be Equal | ${value} | myvalue | | |

catenate (*items)

Catenates the given items together and returns the resulted string.

By default, items are catenated with spaces, but if the first item contains the string ‘SEPARATOR=<sep>’, the separator ‘<sep>’ is used. Items are converted into strings when necessary.

Examples:

| ${str1} = | Catenate | Hello | world | | | | ${str2} = | Catenate | SEPARATOR=— | Hello | world | | | | | ${str3} = | Catenate | SEPARATOR= | Hello | world | => | ${str1} = 'Hello world' | ${str2} = 'Hello—world' | ${str3} = 'Helloworld'

comment (*messages)

Displays the given messages in the log file as keyword arguments.

This keyword does nothing with the arguments it receives, but as they are visible in the log, this keyword can be used to display simple messages. Given arguments are ignored so thoroughly that they can even contain non-existing variables. If you are interested about variable values, you can use the Log or Log Many keywords.

convert_to_binary (item, base=None, prefix=None, length=None)

Converts the given item to a binary string.

The item, with an optional base, is first converted to an integer using Convert To Integer internally. After that it is converted to a binary number (base 2) represented as a string such as ‘1011’.

The returned value can contain an optional prefix and can be required to be of minimum length (excluding the prefix and a possible minus sign). If the value is initially shorter than the required length, it is padded with zeros.

Examples:

| ${result} = | Convert To Binary | 10 | # Result is 1010 | | | | ${result} = | Convert To Binary | F | base=16 | prefix=0b | # Result is 0b1111 | | | | | | length=4 | # Result is -B0010 |

This keyword was added in Robot Framework 2.6. See also Convert To Integer, Convert To Octal and Convert To Hex.

convert_to_boolean (item)

Converts the given item to Boolean true or false.

Handles strings ‘True’ and ‘False’ (case-insensitive) as expected, otherwise returns item’s truth value using Python’s ‘bool’ method. For more information about truth values, see [http://docs.python.org/lib/truth.html](http://docs.python.org/lib/truth.html).

convert_to_hex (item, base=None, prefix=None, length=None, lowercase=False)

Converts the given item to a hexadecimal string.
The `item`, with an optional `base`, is first converted to an integer using `Convert To Integer` internally. After
that it is converted to a hexadecimal number (base 16) represented as a string such as ‘FF0A’.

The returned value can contain an optional `prefix` and can be required to be of minimum `length` (excluding
the prefix and a possible minus sign). If the value is initially shorter than the required length, it is padded
with zeros.

By default the value is returned as an upper case string, but giving any non-empty value to the `lowercase`
argument turns the value (but not the prefix) to lower case.

Examples:

| ${result} = | Convert To Hex | 255 | | # Result is FF |
| ${result} = | Convert To Hex | -10 | prefix=0x | length=2 | # Result is -0x0A |
| ${result} = | Convert To Hex | 255 | prefix=X | lowercase=yes | # Result is Xff |

This keyword was added in Robot Framework 2.6. See also `Convert To Integer`, `Convert To Binary` and
`Convert To Octal`.

### convert_to_integer (item, base=None)

Converts the given item to an integer number.

If the given item is a string, it is by default expected to be an integer in base 10. Starting from Robot
Framework 2.6 there are two ways to convert from other bases:

1. Give base explicitly to the keyword as `base` argument.

2) Prefix the given string with the base so that `0b` means binary (base 2), `0o` means octal (base 8), and `0x`
means hex (base 16). The prefix is considered only when `base` argument is not given and may itself be
prefixed with a plus or minus sign.

The syntax is case-insensitive and possible spaces are ignored.

Examples:

| ${result} = | Convert To Integer | 100 | | # Result is 100 |
| ${result} = | Convert To Integer | FF AA | 16 | # Result is 65450 |
| ${result} = | Convert To Integer | 100 | 8 | # Result is 64 |
| ${result} = | Convert To Integer | -100 | 2 | # Result is -4 |
| ${result} = | Convert To Integer | 0b100 | | # Result is 4 |
| ${result} = | Convert To Integer | -0x100 | | # Result is -256 |

See also `Convert To Number`, `Convert To Binary`, `Convert To Octal` and `Convert To Hex`.

### convert_to_number (item, precision=None)

Converts the given item to a floating point number.

If the optional `precision` is positive or zero, the returned number is rounded to that number of decimal
digits. Negative precision means that the number is rounded to the closest multiple of 10 to the power of
the absolute precision. The support for precision was added in Robot Framework 2.6.

Examples:

| ${result} = | Convert To Number | 42.512 | | # Result is 42.512 |
| ${result} = | Convert To Number | 42.512 | 1 | # Result is 42.5 |
| ${result} = | Convert To Number | 42.512 | 0 | # Result is 43.0 |
| ${result} = | Convert To Number | 42.512 | -1 | # Result is 40.0 |

Notice that machines generally cannot store floating point numbers accurately. This may cause surprises
with these numbers in general and also when they are rounded. For more information see, for example,
this floating point arithmetic tutorial: http://docs.python.org/tutorial/floatingpoint.html

If you need an integer number, use `Convert To Integer` instead.

### convert_to_octal (item, base=None, prefix=None, length=None)

Converts the given item to an octal string.

The `item`, with an optional `base`, is first converted to an integer using `Convert To Integer` internally. After
that it is converted to an octal number (base 8) represented as a string such as ‘775’.
The returned value can contain an optional prefix and can be required to be of minimum length (excluding the prefix and a possible minus sign). If the value is initially shorter than the required length, it is padded with zeros.

Examples: 1 $\{result\} = \text{Convert To Octal} \ 10 \ \# \ \text{Result is} \ 12 \ \$\{result\} = \text{Convert To Octal} -F \ 1 \ \text{base} = 16 \ \$\{prefix\} = 0 \ \# \ \text{Result is} -017 \ \$\{result\} = \text{Convert To Octal} 16 \ \$\{prefix\} = \text{oct} \ \text{length} = 4 \ \# \ \text{Result is oct0020} 1

This keyword was added in Robot Framework 2.6. See also Convert To Integer, Convert To Binary and Convert To Hex.

**convert_to_string** *(item)*

Converts the given item to a Unicode string.

Uses '__unicode__' or '__str__' method with Python objects and 'toString' with Java objects.

Use Encode String To Bytes and Decode Bytes To String keywords in String library if you need to convert between Unicode and byte strings.

**create_list** *(items)*

Returns a list containing given items.

The returned list can be assigned both to $\{scalar\} and @\{list\} variables. The earlier can be used e.g. with Java keywords expecting an array as an argument.

Examples: 1 @\{list\} = \text{Create List} a b c \ \$\{scalar\} = \text{Create List} a b c \ \$\{ints\} = \text{Create List} $\{1\} 2 $\{3\}

**evaluate** *(expression, modules=None)*

Evaluates the given expression in Python and returns the results.

*modules* argument can be used to specify a comma separated list of Python modules to be imported and added to the namespace of the evaluated expression.

Examples (expecting $\{result\} is 3.14): 1 $\{status\} = \text{Evaluate} 0 < $\{result\} < 10 \ 11 $\{down\} = \text{Evaluate} \ \text{int}($\{result\}) \ 11 $\{up\} = \text{Evaluate} 1 \ \text{math.ceil}($\{result\}) \ \text{math} 11 $\{random\} = 1 \ \text{Evaluate} 1 \ \text{random.randint}(0, \ \text{sys.maxint}) \ \text{random,sys} \Rightarrow 1 \ \text{status} = \text{True} 1 \ \text{down} = 3 \ 1 \ \text{up} = 4.0 \ 1 \ \text{random} = \text{<random integer>}

Notice that instead of creating complicated expressions, it is recommended to move the logic into a test library.

**exit_for_loop** *

Immediately stops executing the enclosing for loop.

This keyword can be used directly in a for loop or in a keyword that the for loop uses. In both cases the test execution continues after the for loop. If executed outside of a for loop, the test fails.

Example: 1 :FOR \ $\{var\} \ IN \ @\{SOME LIST\} \ | | Run Keyword If \ $\{var\}' == 'EXIT' \ Exit For Loop \ | | Do Something \ $\{var\} \ |

New in Robot Framework 2.5.2.

**fail** *(msg=None, *tags)*

Fails the test with the given message and optionally alters its tags.

The error message is specified using the optional *msg* argument.

Starting from Robot Framework 2.7.4, it is possible to modify tags of the current test case by passing tags after the message. Tags starting with a hyphen (e.g. -regression) are removed and others added. Tags are modified using Set Tags and Remove Tags internally, and the semantics setting and removing them are the same as with these keywords.
Examples: | Fail | Keyword not ready | # Fails with the given message. | Fail | Keyword not ready | not-ready | # Fails and adds ‘not-ready’ tag. | Fail | OS not supported | -regression | # Removes tag ‘regression’. | Fail | My message | -old | new | # Adds tag ‘new’ and removes ‘old’. | Fail | My message | tag | -t* | # Removes all tags starting with ‘t’ except the newly added ‘tag’. |

See Fatal Error if you need to stop the whole test execution.

### fatal_error (msg=None)

Stops the whole test execution.

The test or suite where this keyword is used fails with the provided message, and subsequent tests fail with a canned message. Possible teardowns will nevertheless be executed.

See Fail if you only want to stop one test case unconditionally.

### get_count (item1, item2)

Returns and logs how many times `item2` is found from `item1`.

This keyword works with Python strings and lists and all objects that either have ‘count’ method or can be converted to Python lists.

Example: | ${count} = | Get Count | ${some item} | interesting value | Should Be True | 5 < ${count} < 10 |

### get_length (item)

Returns and logs the length of the given item.

The item can be anything that has a length, for example, a string, a list, or a mapping. The keyword first tries to get the length with the Python function `len`, which calls the item’s `__len__` method internally. If that fails, the keyword tries to call the item’s possible `length` and `size` methods directly. The final attempt is trying to get the value of the item’s `length` attribute. If all these attempts are unsuccessful, the keyword fails.

It is possible to use this keyword also with list variables (e.g. `@{LIST}`), but you need to use them as scalars (e.g. `$LIST`).

### get_library_instance (name)

Returns the currently active instance of the specified test library.

This keyword makes it easy for test libraries to interact with other test libraries that have state. This is illustrated by the Python example below:

```python
from robot.libraries.BuiltIn import BuiltIn

def title_should_start_with(expected):
    seleniumlib = BuiltIn().get_library_instance('SeleniumLibrary')
    title = seleniumlib.get_title()
    if not title.startswith(expected):
        raise AssertionError("Title ‘%s’ did not start with ‘%s’
                         % (title, expected))

It is also possible to use this keyword in the test data and pass the returned library instance to another keyword. If a library is imported with a custom name, the `name` used to get the instance must be that name and not the original library name.

### get_time (format='timestamp', time_='NOW')

Returns the given time in the requested format.
How time is returned is determined based on the given `format` string as follows. Note that all checks are case-insensitive.

1. If `format` contains the word ‘epoch’, the time is returned in seconds after the UNIX epoch (1970-01-01 00:00:00 UTC). The return value is always an integer.

2. If `format` contains any of the words ‘year’, ‘month’, ‘day’, ‘hour’, ‘min’, or ‘sec’, only the selected parts are returned. The order of the returned parts is always the one in the previous sentence and the order of words in `format` is not significant. The parts are returned as zero-padded strings (e.g. May -> ‘05’).

3. Otherwise (and by default) the time is returned as a timestamp string in the format ‘2006-02-24 15:08:31’.

By default this keyword returns the current local time, but that can be altered using `time` argument as explained below. Note that all checks involving strings are case-insensitive.

1. If `time` is a number, or a string that can be converted to a number, it is interpreted as seconds since the UNIX epoch. This documentation was originally written about 1177654467 seconds after the epoch.

2. If `time` is a timestamp, that time will be used. Valid timestamp formats are ‘YYYY-MM-DD hh:mm:ss’ and ‘YYYYMMDD hh:mm:ss’.

3. If `time` is equal to ‘NOW’ (default), the current local time is used. This time is got using Python’s `time.time()` function.

4. If `time` is equal to ‘UTC’, the current time in [http://en.wikipedia.org/wiki/Coordinated_Universal_Time|UTC] is used. This time is got using ‘time.time() + time.altzone’ in Python.

5. If `time` is in the format like ‘NOW - 1 day’ or ‘UTC + 1 hour 30 min’, the current local/UTC time plus/minus the time specified with the time string is used. The time string format is described in an appendix of Robot Framework User Guide.

Examples (expecting the current local time is 2006-03-29 15:06:21): | ${time} = | Get Time | | | | | ${secs} = | Get Time | sec | 2007-04-27 09:14:27 |
| => | ${time} = '2007-04-27 09:14:27' |

Examples (expecting the current local time is 2006-03-29 15:06:21 and UTC time is 2006-03-29 12:06:21): | ${time} = | Get Time | | 1177654467 | # Time given as epoch seconds |
| => | ${time} = 1177654467 |
| | $\{time\} = | Get Time | | | | 1177654467 | | # Time given as epoch seconds |
| => | $\{time\} = 1177654467 |

Support for UTC time was added in Robot Framework 2.7.5 but it did not work correctly until 2.7.7.

`get_variable_value(name, default=None)`

Returns variable value or `default` if the variable does not exist.

The name of the variable can be given either as a normal variable name (e.g. `$\{NAME\}`) or in escaped format (e.g. `$NAME`). Notice that the former has some limitations explained in Set Suite Variable.

Examples: | ${x} = | Get Variable Value | | | | | ${a} = | Get Variable Value | | | | | ${y} = | Get Variable Value | | => | ${x} gets value of ${a} if ${a} exists and string "default" otherwise | $\{z\} gets value of ${a} if ${a} exists and value of ${b} otherwise | $\{z\} is set to Python None if it does not exist previously.
This keyword was added in Robot Framework 2.6. See Set Variable If for another keyword to set variables dynamically.

**get_variables()**

Returns a dictionary containing all variables in the current scope.

Variables are returned as a special dictionary that allows accessing variables in space, case, and underscore insensitive manner similarly as accessing variables in the test data. This dictionary supports all same operations as normal Python dictionaries and, for example, Collections library can be used to access or modify it. Modifying the returned dictionary has no effect on the variables available in the current scope.

Example:
```
$example_variable = Set Variable|example value|
$variables = Get Variables
```

Note: Prior to Robot Framework 2.7.4 variables were returned as a custom object that did not support all dictionary methods.

**import_library(name, *args)**

Imports a library with the given name and optional arguments.

This functionality allows dynamic importing of libraries while tests are running. That may be necessary, if the library itself is dynamic and not yet available when test data is processed. In a normal case, libraries should be imported using the Library setting in the Setting table.

This keyword supports importing libraries both using library names and physical paths. When path are used, they must be given in absolute format. Forward slashes can be used as path separators in all operating systems. It is possible to use arguments as well as to give a custom name with ‘WITH NAME’ syntax. For more information about importing libraries, see Robot Framework User Guide.

Examples:
```
Import Library|MyLibrary|
Import Library|${CURDIR}/Library.py|some|args|
Import Library|${CURDIR}/../libs/Lib.java|arg|WITH NAME|JavaLib|
```

**import_resource(path)**

Imports a resource file with the given path.

Resources imported with this keyword are set into the test suite scope similarly when importing them in the Setting table using the Resource setting.

The given path must be absolute. Forward slashes can be used as path separator regardless the operating system.

Examples:
```
Import Resource|${CURDIR}/resource.txt|
Import Resource|${CURDIR}/../resources/resource.html|
```

**import_variables(path, *args)**

Imports a variable file with the given path and optional arguments.

Variables imported with this keyword are set into the test suite scope similarly when importing them in the Setting table using the Variables setting. These variables override possible existing variables with the same names and this functionality can thus be used to import new variables, e.g. for each test in a test suite.

The given path must be absolute. Forward slashes can be used as path separator regardless the operating system.

Examples:
```
Import Variables|${CURDIR}/variables.py|
Import Variables|${CURDIR}/..vars/env.py|arg1|arg2|
```

New in Robot Framework 2.5.4.

**keyword_should_exist(name, msg=None)**

Fails unless the given keyword exists in the current scope.
Fails also if there are more than one keywords with the same name. Works both with the short name (e.g. *Log*) and the full name (e.g. *BuiltIn.Log*).

The default error message can be overridden with the *msg* argument.

New in Robot Framework 2.6. See also Variable Should Exist.

**length_should_be** (*item, length, msg=None*)

Verifies that the length of the given item is correct.

The length of the item is got using the *Get Length* keyword. The default error message can be overridden with the *msg* argument.

**log** (*message, level='INFO'*)

Logs the given message with the given level.

Valid levels are TRACE, DEBUG, INFO (default), HTML and WARN.

The HTML level is special because it allows writing messages without HTML code in them being escaped. For example, logging a message `<img src="image.png">` using the HTML level creates an image, but with other levels the message would be that exact string. Notice that invalid HTML can easily corrupt the whole log file so this feature should be used with care. The actual log level used for HTML messages is INFO.

Messages logged with the WARN level will be visible also in the console and in the Test Execution Errors section in the log file.

**log_many** (*messages*)

Logs the given messages as separate entries with the INFO level.

**log_variables** (level='INFO')

Logs all variables in the current scope with given log level.

**no_operation** ()

Does absolutely nothing.

**regexp_escape** (*patterns*)

Returns each argument string escaped for use as a regular expression.

This keyword can be used to escape strings to be used with *Should Match Regexp* and *Should Not Match Regexp* keywords.

Escaping is done with Python’s re.escape() function.

Examples: | ${escaped} = | Regexp Escape | ${original} | | @{strings} = | Regexp Escape | @{strings} |

**remove_tags** (*tags*)

Removes given *tags* from the current test or all tests in a suite.

Tags can be given exactly or using a pattern where ‘*’ matches anything and ‘?’ matches one character.

This keyword can affect either one test case or all test cases in a test suite similarly as *Set Tags* keyword.

The current tags are available as a built-in variable `@{TEST TAGS}`.

Example: | Remove Tags | mytag | something-* | ?ython |

See *Set Tags* if you want to add certain tags and *Fail* if you want to fail the test case after setting and/or removing tags.

**repeat_keyword** (times, name, *args*)

Executes the specified keyword multiple times.

*name* and *args* define the keyword that is executed similarly as with *Run Keyword*, and *times* specifies how many the keyword should be executed. *times* can be given as an integer or as a string that can be converted
to an integer. It can also have postfix ‘times’ or ‘x’ (case and space insensitive) to make the expression
easier to read.

If times is zero or negative, the keyword is not executed at all. This keyword fails immediately if any of
the execution rounds fails.

Examples: | Repeat Keyword | 5 times | Goto Previous Page | | Repeat Keyword | ${var} | Some Keyword
| arg1 | arg2 |

replace_variables (text)
Replaces variables in the given text with their current values.

If the text contains undefined variables, this keyword fails. If the given text contains only a single variable,
its value is returned as-is and it can be any object. Otherwise this keyword always returns a string.

Example:
The file ‘template.txt’ contains ‘Hello ${NAME}!’ and variable ‘${NAME}’ has the value ‘Robot’.

${template} = | Get File | ${CURDIR}/template.txt |
${message} = | Replace Variables | ${template} |
Should Be Equal | ${message} | Hello Robot! |

run_keyword (name, *args)
Executes the given keyword with the given arguments.

Because the name of the keyword to execute is given as an argument, it can be a variable and thus set
dynamically, e.g. from a return value of another keyword or from the command line.

run_keyword_and_continue_on_failure (name, *args)
Runs the keyword and continues execution even if a failure occurs.

The keyword name and arguments work as with Run Keyword.

Example: | Run Keyword And Continue On Failure | Fail | This is a stupid example | | Log | This keyword
is executed |

This keyword was added in Robot Framework 2.5. The execution is not continued if the failure is caused
by invalid syntax, timeout, or fatal exception.

run_keyword_and_expect_error (expected_error, name, *args)
Runs the keyword and checks that the expected error occurred.

The expected error must be given in the same format as in Robot Framework reports. It can be a pattern
containing characters ‘?’, which matches to any single character and ‘*’, which matches to any number of
any characters. name and *args have same semantics as with Run Keyword.

If the expected error occurs, the error message is returned and it can be further processed/tested, if needed.
If there is no error, or the error does not match the expected error, this keyword fails.

Examples: | Run Keyword And Expect Error | My error | Some Keyword | arg1 | arg2 | | ${msg} = | Run Keyword And Expect Error | * | My KW | | Should Start With | ${msg} | Once upon a time in |

Starting from Robot Framework 2.5 errors caused by invalid syntax, timeouts, or fatal exceptions are not
captured by this keyword.

run_keyword_and_ignore_error (name, *args)
Runs the given keyword with the given arguments and ignores possible error.
This keyword returns two values, so that the first is either ‘PASS’ or ‘FAIL’, depending on the status of the executed keyword. The second value is either the return value of the keyword or the received error message. See Run Keyword And Return Status If you are only interested in the execution status.

The keyword name and arguments work as in Run Keyword. See Run Keyword If for a usage example.

Starting from Robot Framework 2.5 errors caused by invalid syntax, timeouts, or fatal exceptions are not caught by this keyword.

\texttt{run\_keyword\_and\_return\_status}(\texttt{name}, \texttt{\*args})

Runs the given keyword with given arguments and returns the status as a Boolean value.

This keyword returns \texttt{True} if the keyword that is executed succeeds and \texttt{False} if it fails. This is useful, for example, in combination with Run Keyword If. If you are interested in the error message or return value, use Run Keyword And Ignore Error instead.

The keyword name and arguments work as in Run Keyword.

Example: \texttt{\{passed\} = Run Keyword And Return Status | Keyword \{args\} | Run Keyword If \{\{passed\}\} | Another keyword} \n
New in Robot Framework 2.7.6.

\texttt{run\_keyword\_if}(\texttt{condition}, \texttt{name}, \texttt{\*args})

Runs the given keyword with the given arguments, if \texttt{condition} is true.

The given \texttt{condition} is evaluated similarly as with Should Be True keyword, and \texttt{name} and \texttt{\*args} have same semantics as with Run Keyword.

Example, a simple if/else construct: \texttt{\{status\} \{value\} = Run Keyword And Ignore Error | My Keyword | Run Keyword If \{\{status\}\} == ‘PASS’ | Some Action \{arg\} | Run Keyword Unless \{\{status\}\} == ‘PASS’ | Another Action} \n
In this example, only either \texttt{Some Action} or \texttt{Another Action} is executed, based on the status of \texttt{My Keyword}. Instead of Run Keyword And Ignore Error you can also use Run Keyword And Return Status.

Starting from Robot version 2.7.4, this keyword supports also optional ELSE and ELSE IF branches. Both of these are defined in \texttt{\*args} and must use exactly format \texttt{ELSE} or \texttt{ELSE IF}, respectively. ELSE branches must contain first the name of the keyword to execute and then its possible arguments. ELSE IF branches must first contain a condition, like the first argument to this keyword, and then the keyword to execute and its possible arguments. It is possible to have ELSE branch after ELSE IF and to have multiple ELSE IF branches.

Given previous example, if/else construct can also be created like this: \texttt{\{status\} \{value\} = Run Keyword And Ignore Error | My Keyword | Run Keyword If \{\{status\}\} == ‘PASS’ | Some Action \{arg\} | ELSE | Another Action} \n
Using ELSE and/or ELSE IF branches is especially handy if you are interested in the return value. This is illustrated by the example below that also demonstrates using ELSE IF and ELSE together:

\texttt{\{result\} = Run Keyword If \{\{rc\}\} == 0 | Zero return value | ELSE IF \{\{rc\}\} < 42 | Normal return value | ELSE IF \{\{rc\}\} < 0 | Negative return value \{\{rc\}\} \{arg2\} | ELSE | Abnormal return value \{\{rc\}\} |}

Notice that ELSE and ELSE IF control arguments must be used explicitly and thus cannot come from variables. If you need to use literal ELSE and ELSE IF strings as arguments, you can either use variables or escape them with a backslash like \texttt{ELSE} and \texttt{ELSE IF}. 

254 Chapter 4. All packages
**run_keyword_if_all_critical_tests_passed** (name, *args)
Runs the given keyword with the given arguments, if all critical tests passed.

This keyword can only be used in suite teardown. Trying to use it in any other place will result in an error.

Otherwise, this keyword works exactly like `Run Keyword`, see its documentation for more details.

**run_keyword_if_all_tests_passed** (name, *args)
Runs the given keyword with the given arguments, if all tests passed.

This keyword can only be used in a suite teardown. Trying to use it anywhere else results in an error.

Otherwise, this keyword works exactly like `Run Keyword`, see its documentation for more details.

**run_keyword_if_any_critical_tests_failed** (name, *args)
Runs the given keyword with the given arguments, if any critical tests failed.

This keyword can only be used in a suite teardown. Trying to use it anywhere else results in an error.

Otherwise, this keyword works exactly like `Run Keyword`, see its documentation for more details.

**run_keyword_if_any_tests_failed** (name, *args)
Runs the given keyword with the given arguments, if one or more tests failed.

This keyword can only be used in a suite teardown. Trying to use it anywhere else results in an error.

Otherwise, this keyword works exactly like `Run Keyword`, see its documentation for more details.

**run_keyword_if_test_failed** (name, *args)
Runs the given keyword with the given arguments, if the test failed.

This keyword can only be used in a test teardown. Trying to use it anywhere else results in an error.

Otherwise, this keyword works exactly like `Run Keyword`, see its documentation for more details.

**run_keyword_if_test_passed** (name, *args)
Runs the given keyword with the given arguments, if the test passed.

This keyword can only be used in a test teardown. Trying to use it anywhere else results in an error.

Otherwise, this keyword works exactly like `Run Keyword`, see its documentation for more details.

**run_keyword_if_timeout_occurred** (name, *args)
Runs the given keyword if either a test or a keyword timeout has occurred.

This keyword can only be used in a test teardown. Trying to use it anywhere else results in an error.

Otherwise, this keyword works exactly like `Run Keyword`, see its documentation for more details.

Available in Robot Framework 2.5 and newer.

**run_keyword_unless** (condition, name, *args)
Runs the given keyword with the given arguments, if condition is false.

See `Run Keyword If` for more information and an example.

**run_keywords** (*keywords)
Executes all the given keywords in a sequence.

This keyword is mainly useful in setups and tear downs when they need to take care of multiple actions and creating a new higher level user keyword would be an overkill.

Examples: | Run Keywords | Initialize database | Start servers | Clear logs | | Run Keywords | ${KW 1} | ${KW 2} | | Run Keywords | @{KEYWORDS} |

In this example, we call `Run Keywords` with three different combination of arguments. Keyword names and arguments can come from variables, as demonstrated in the second and third row.
Starting from Robot Framework 2.7.6, keywords can also be run with arguments using upper case `AND` as a separator between keywords. The keywords are executed so that the first argument is the first keyword and proceeding arguments until the first `AND` are arguments to it. First argument after the first `AND` is the second keyword and proceeding arguments until the next `AND` are its arguments. And so on.

Examples:
```
| Run Keywords | Initialize database | db1 | AND | Start servers | server1 | server2 |
| Run Keywords | Initialize database | ${DB NAME} | AND | Start servers | @{SERVERS} | AND | Clear logs |
| Run Keywords | ${KW} | AND | @{KW WITH ARGS} |
```

Notice that the `AND` control argument must be used explicitly and thus cannot itself come from a variable. If you need to use literal `AND` string as argument, you can either use variables or escape it with a backslash like `\AND`.

### set_global_variable
```
(name, *values)
```

Makes a variable available globally in all tests and suites.

Variables set with this keyword are globally available in all test cases and suites executed after setting them. Setting variables with this keyword thus has the same effect as creating from the command line using the options `--variable` or `--variablefile`. Because this keyword can change variables everywhere, it should be used with care.

See Set Suite Variable for more information and examples.

### set_library_search_order
```
(*libraries)
```

Sets the resolution order to use when a name matches multiple keywords.

The library search order is used to resolve conflicts when a keyword name in the test data matches multiple keywords. The first library (or resource, see below) containing the keyword is selected and that keyword implementation used. If the keyword is not found from any library (or resource), test executing fails the same way as when the search order is not set.

When this keyword is used, there is no need to use the long `LibraryName.Keyword Name` notation. For example, instead of having

```
MyLibrary.Keyword | arg | MyLibrary.Another Keyword | MyLibrary.Keyword | xxx |
```

you can have

```
Set Library Search Order | MyLibrary |
Keyword | arg |
Another Keyword |
Keyword | xxx |
```

Starting from Robot Framework 2.6.2 this keyword can be used also to set the order of keywords in different resource files. In this case resource names must be given without paths or extensions like:

```
Set Library Search Order | resource | another_resource |
```
NOTE: - The search order is valid only in the suite where this keywords is used. - Keywords in resources always have higher priority than
  keywords in libraries regardless the search order.

• The old order is returned and can be used to reset the search order later.
• Starting from RF 2.6.2, library and resource names in the search order are both case and space insensitive.

**set_log_level** *(level)*
Sets the log threshold to the specified level and returns the old level.

Messages below the level will not logged. The default logging level is INFO, but it can be overridden with the command line option `–loglevel`.

The available levels: TRACE, DEBUG, INFO (default), WARN and NONE (no logging).

**set_suite_documentation** *(doc, append=False, top=False)*
Sets documentation for the current test suite.

By default the possible existing documentation is overwritten, but this can be changed using the optional `append` argument similarly as with `Set Test Message` keyword.

This keyword sets the documentation of the current suite by default. If the optional `top` argument is given any value considered `true` in Python, for example, any non-empty string, the documentation of the top level suite is altered instead.

The documentation of the current suite is available as a built-in variable `$[SUITE DOCUMENTATION]`.

New in Robot Framework 2.7. Support for `append` and `top` were added in 2.7.7.

**set_suite_metadata** *(name, value, append=False, top=False)*
Sets metadata for the current test suite.

By default possible existing metadata values are overwritten, but this can be changed using the optional `append` argument similarly as with `Set Test Message` keyword.

This keyword sets the metadata of the current suite by default. If the optional `top` argument is given any value considered `true` in Python, for example, any non-empty string, the metadata of the top level suite is altered instead.

The metadata of the current suite is available as a built-in variable `$[SUITE METADATA]` in a Python dictionary. Notice that modifying this variable directly has no effect on the actual metadata the suite has.

New in Robot Framework 2.7.4. Support for `append` and `top` were added in 2.7.7.

**set_suite_variable** *(name, *values)*
Makes a variable available everywhere within the scope of the current suite.

Variables set with this keyword are available everywhere within the scope of the currently executed test suite. Setting variables with this keyword thus has the same effect as creating them using the Variable table in the test data file or importing them from variable files. Other test suites, including possible child test suites, will not see variables set with this keyword.

The name of the variable can be given either as a normal variable name (e.g. `$[NAME]`) or in escaped format as `@$NAME` or `$NAME`.

If a variable already exists within the new scope, its value will be overwritten. Otherwise a new variable is created. If a variable already exists within the current scope, the value can be left empty and the variable within the new scope gets the value within the current scope.
Examples:
| Set Suite Variable | ${GREET} | Hello, world! |
| Set Suite Variable | @ {LIST} | First item |
|                   | Second item | Second item |
|                   | $ {ID} | Get ID |
| Set Suite Variable | $ {ID} |

To override an existing value with an empty value, use built-in variables $\{EMPTY\}$ or @\{EMPTY\}:

Set Suite Variable | $ {GREET} | $ {EMPTY} |
|                   | @ {LIST} | @ {EMPTY} | \# New in RF 2.7.4 |

**NOTE:** If the variable has value which itself is a variable (escaped or not), you must always use the escaped format to reset the variable:

Example:

| Set Variable | $ {var} || Set Suite Variable | $ {NAME} | value || # Sets variable $ {var} || Set Suite Variable | $ {NAME} | value || # Sets variable $ {NAME} |

This limitation applies also to Set Test/Suite/Global Variable, Variable Should (Not) Exist, and Get Variable Value keywords.

**set_tags** (*tags*)

Adds given *tags* for the current test or all tests in a suite.

When this keyword is used inside a test case, that test gets the specified tags and other tests are not affected.

If this keyword is used in a suite setup, all test cases in that suite, recursively, gets the given tags. It is a failure to use this keyword in a suite teardown.

The current tags are available as a built-in variable @\{TEST TAGS\}.

See Remove Tags if you want to remove certain tags and Fail if you want to fail the test case after setting and/or removing tags.

**set_test_documentation** (*doc*, append=False)

Sets documentation for the current test case.

By default the possible existing documentation is overwritten, but this can be changed using the optional *append* argument similarly as with Set Test Message keyword.

The current test documentation is available as a built-in variable $\{TEST DOCUMENTATION\}$. This keyword can not be used in suite setup or suite teardown.

New in Robot Framework 2.7. Support for *append* was added in 2.7.7.

**set_test_message** (*message*, append=False)

Sets message for the current test case.

If this keyword is used outside a test teardown, the message is overridden by possible failure message. If this is used in teardown, possible earlier failure message is overridden. Failures in teardown are always shown in addition to this message.

If the optional *append* argument is given any value considered *true* in Python, for example, any non-empty string, the given *message* is added after the possible earlier message by joining the messages with a space.

In teardown the current test message is available as a built-in variable $\{TEST MESSAGE\}$. This keyword can not be used in suite setup or suite teardown.

Examples:

| Set Test Message | My message | is continued. | append=yes || Should Be Equal | $ {TEST MESSAGE} | My message is continued. |

New in Robot Framework 2.5. Support for *append* was added in 2.7.7.

**set_test_variable** (*name*, *values*)

Makes a variable available everywhere within the scope of the current test.
Variables set with this keyword are available everywhere within the scope of the currently executed test case. For example, if you set a variable in a user keyword, it is available both in the test case level and also in all other user keywords used in the current test. Other test cases will not see variables set with this keyword.

See Set Suite Variable for more information and examples.

**set_variable** (*values*)

Returns the given values which can then be assigned to a variables.

This keyword is mainly used for setting scalar variables. Additionally it can be used for converting a scalar variable containing a list to a list variable or to multiple scalar variables. It is recommended to use Create List when creating new lists.

Examples: | ${hi} = | Set Variable | Hello, world! | | ${hi2} = | Set Variable | I said: | ${hi} | | ${var1} = | Set Variable | Hello | | ${var2} = | Set Variable | world | | @{list} = | Set Variable | ${list with some items} | | ${item1} = | Set Variable | ${list with 2 items} | | ${item2} = | Set Variable |

Variables created with this keyword are available only in the scope where they are created. See Set Global Variable, Set Test Variable and Set Suite Variable for information on how to set variables so that they are available also in a larger scope.

**set_variable_if** (condition, *values*)

Sets variable based on the given condition.

The basic usage is giving a condition and two values. The given condition is first evaluated the same way as with the Should Be True keyword. If the condition is true, then the first value is returned, and otherwise the second value is returned. The second value can also be omitted, in which case it has a default value None. This usage is illustrated in the examples below, where $\{rc\}$ is assumed to be zero.

${var1} = | Set Variable If | $\{rc\} == 0 | zero | nonzero |
${var2} = | Set Variable If | $\{rc\} > 0 | value1 | value2 |
${var3} = | Set Variable If | $\{rc\} > 0 | whatever |

Rightarrow | ${var1} = ‘zero’ | ${var2} = ‘value2’ | ${var3} = None

It is also possible to have ‘Else If’ support by replacing the second value with another condition, and having two new values after it. If the first condition is not true, the second is evaluated and one of the values after it is returned based on its truth value. This can be continued by adding more conditions without a limit.

$\{var\} = | Set Variable If | $\{rc\} == 0 | zero |
| $\{rc\} > 0 | greater than zero | less then zero |

$\{var\} = | Set Variable If |
| $\{rc\} == 0 | zero |
| $\{rc\} == 1 | one |
| $\{rc\} == 2 | two |
| $\{rc\} > 2 | greater than two |
| $\{rc\} < 0 | less than zero |

Use Get Variable Value if you need to set variables dynamically based on whether a variable exist or not.
**should_be_empty**(item, msg=None)

Verifies that the given item is empty.

The length of the item is got using the Get Length keyword. The default error message can be overridden with the msg argument.

**should_be_equal**(first, second, msg=None, values=True)

Fails if the given objects are unequal.

- If msg is not given, the error message is ‘first != second’.
- If msg is given and values is either Boolean False or the string ‘False’ or ‘No Values’, the error message is simply msg.
- Otherwise the error message is ‘msg: first != second’.

**should_be_equal_as_integers**(first, second, msg=None, values=True, base=None)

Fails if objects are unequal after converting them to integers.

See Convert To Integer for information how to convert integers from other bases than 10 using base argument or 0b/0o/0x prefixes.

See Should Be Equal for an explanation on how to override the default error message with msg and values.

Examples:

| Should Be Equal As Integers | 42 | ${42} | Error message || Should Be Equal As Integers | ABCD | abcd | base=16 | Should Be Equal As Integers | 0b1011 | 11 |

**should_be_equal_as_numbers**(first, second, msg=None, values=True, precision=6)

Fails if objects are unequal after converting them to real numbers.

The conversion is done with Convert To Number keyword using the given precision. The support for giving precision was added in Robot Framework 2.6, in earlier versions it was hard-coded to 6.

Examples:

| Should Be Equal As Numbers | ${x} | 1.1 | # Passes if ${x} is 1.1 | Should Be Equal As Numbers | 1.123 | 1.1 | precision=1 | # Passes | Should Be Equal As Numbers | 1.123 | 1.4 | precision=0 | # Passes | Should Be Equal As Numbers | 112.3 | 175 | precision=-2 | # Passes |

As discussed in the documentation of Convert To Number, machines generally cannot store floating point numbers accurately. Because of this limitation, comparing floats for equality is problematic and a correct approach to use depends on the context. This keyword uses a very naive approach of rounding the numbers before comparing them, which is both prone to rounding errors and does not work very well if numbers are really big or small. For more information about comparing floats, and ideas on how to implement your own context specific comparison algorithm, see this great article: [http://www.cygnssoft.com/papers/comparingfloats/comparingfloats.htm](http://www.cygnssoft.com/papers/comparingfloats/comparingfloats.htm)

See Should Not Be Equal As Numbers for a negative version of this keyword and Should Be Equal for an explanation on how to override the default error message with msg and values.

**should_be_equal_as_strings**(first, second, msg=None, values=True)

Fails if objects are unequal after converting them to strings.

See Should Be Equal for an explanation on how to override the default error message with msg and values.

**should_be_true**(condition, msg=None)

Fails if the given condition is not true.

If condition is a string (e.g. ‘${rc} < 10’), it is evaluated as a Python expression using the built-in ‘eval’ function and the keyword status is decided based on the result. If a non-string item is given, the status is got directly from its truth value as explained at [http://docs.python.org/lib/truth.html](http://docs.python.org/lib/truth.html).

The default error message (‘<condition> should be true’) is not very informative, but it can be overridden with the msg argument.
Examples: | Should Be True | ${rc} < 10 | | Should Be True | ‘${status}’ == ‘PASS’ | # Strings must be quoted | | Should Be True | ${number} | # Passes if ${number} is not zero | | Should Be True | ${list} | # Passes if ${list} is not empty |

should_contain(item1, item2, msg=None, values=True)
Fails if item1 does not contain item2 one or more times.

Works with strings, lists, and anything that supports Python’s ‘in’ keyword. See Should Be Equal for an explanation on how to override the default error message with msg and values.

Examples: | Should Contain | ${output} | PASS | | Should Contain | ${some_list} | value |

should_contain_x_times(item1, item2, count, msg=None)
Fails if item1 does not contain item2 count times.

Works with strings, lists and all objects that Get Count works with. The default error message can be overridden with msg and the actual count is always logged.

Examples: | Should Contain X Times | ${output} | hello | 2 | | Should Contain X Times | ${some list} | value | 3 |

should_end_with(str1, str2, msg=None, values=True)
Fails if the string str1 does not end with the string str2.

See Should Be Equal for an explanation on how to override the default error message with msg and values.

should_match(string, pattern, msg=None, values=True)
Fails unless the given string matches the given pattern.

Pattern matching is similar as matching files in a shell, and it is always case-sensitive. In the pattern, ‘*’ matches to anything and ‘?’ matches to any single character.

See Should Be Equal for an explanation on how to override the default error message with msg and values.

should_match_regexp(string, pattern, msg=None, values=True)
Fails if string does not match pattern as a regular expression.

Regular expression check is done using the Python ‘re’ module, which has a pattern syntax derived from Perl, and thus also very similar to the one in Java. See the following documents for more details about regular expressions in general and Python implementation in particular.

http://docs.python.org/lib/module-re.html
http://www.amk.ca/python/howto/regex/

Things to note about the regexp syntax in Robot Framework test data:

1) Backslash is an escape character in the test data, and possible backslashes in the pattern must thus be escaped with another backslash (e.g. ‘\d\w+’).

2) Strings that may contain special characters, but should be handled as literal strings, can be escaped with the Regex Escape keyword.

3) The given pattern does not need to match the whole string. For example, the pattern ‘ello’ matches the string ‘Hello world!’. If a full match is needed, the ‘^’ and ‘$’ characters can be used to denote the beginning and end of the string, respectively. For example, ‘^ello$’ only matches the exact string ‘ello’.

4) Possible flags altering how the expression is parsed (e.g. re.IGNORECASE, re.MULTILINE) can be set by prefixing the pattern with the ‘(?iLmsux)’ group (e.g. ‘(?i)pattern’). The available flags are ‘IGNORECASE’: ‘i’, ‘MULTILINE’: ‘m’, ‘DOTALL’: ‘s’, ‘VERBOSE’: ‘x’, ‘UNICODE’: ‘u’, and ‘LOCALE’: ‘l’.

4.7. libraries Package 261
If this keyword passes, it returns the portion of the string that matched the pattern. Additionally, the possible captured groups are returned.

See the `Should Be Equal` keyword for an explanation on how to override the default error message with the `msg` and `values` arguments.

Examples: | Should Match Regexp | ${output} | \d{6} | # Output contains six numbers | | Should Match Regexp | ${output} | ^\d{6}$ | # Six numbers and nothing more | | Should Match Regexp | Foo: 42 | (?i)foo: \d+ | | Should Match Regexp | Bar: 43 | (Foo|Bar): (\d+) | => | ${ret} = 'Foo: 42' | ${match} = 'Bar: 43' | ${group1} = 'Bar' | ${group2} = '43'

`should_not_be_empty` (item, msg=None)
Verifies that the given item is not empty.

The length of the item is got using the `Get Length` keyword. The default error message can be overridden with the `msg` argument.

`should_not_be_equal` (first, second, msg=None, values=True)
Fails if the given objects are equal.

See `Should Be Equal` for an explanation on how to override the default error message with `msg` and `values`.

`should_not_be_equal_as_integers` (first, second, msg=None, values=True, base=None)
Fails if objects are equal after converting them to integers.

See `Convert To Integer` for information how to convert integers from other bases than 10 using `base` argument or `0b/0o/0x` prefixes.

See `Should Be Equal` for an explanation on how to override the default error message with `msg` and `values`.

See `Should Be Equal As Integers` for some usage examples.

`should_not_be_equal_as_numbers` (first, second, msg=None, values=True, precision=6)
Fails if objects are equal after converting them to real numbers.

The conversion is done with `Convert To Number` keyword using the given `precision`. The support for giving precision was added in Robot Framework 2.6, in earlier versions it was hard-coded to 6.

See `Should Be Equal As Numbers` for examples on how to use `precision` and why it does not always work as expected. See also `Should Be Equal` for an explanation on how to override the default error message with `msg` and `values`.

`should_not_be_equal_as_strings` (first, second, msg=None, values=True)
Fails if objects are equal after converting them to strings.

See `Should Be Equal` for an explanation on how to override the default error message with `msg` and `values`.

`should_not_be_true` (condition, msg=None)
Fails if the given condition is true.

See `Should Be True` for details about how `condition` is evaluated and how `msg` can be used to override the default error message.

`should_not_contain` (item1, item2, msg=None, values=True)
Fails if `item1` contains `item2` one or more times.

Works with strings, lists, and anything that supports Python’s `in` keyword. See `Should Be Equal` for an explanation on how to override the default error message with `msg` and `values`.

Examples: | Should Not Contain | ${output} | FAILED | | Should Not Contain | ${some_list} | value |

`should_not_end_with` (str1, str2, msg=None, values=True)
Fails if the string `str1` ends with the string `str2`.

See `Should Be Equal` for an explanation on how to override the default error message with `msg` and `values`.
should_not_match (string, pattern, msg=None, values=True)
Fails if the given string matches the given pattern.

Pattern matching is similar as matching files in a shell, and it is always case-sensitive. In the pattern ‘*’ matches to anything and ‘?’ matches to any single character.

See Should Be Equal for an explanation on how to override the default error message with msg and values.

should_not_match_regexp (string, pattern, msg=None, values=True)
Fails if string matches pattern as a regular expression.

See Should Match Regexp for more information about arguments.

should_not_start_with (str1, str2, msg=None, values=True)
Fails if the string str1 starts with the string str2.

See Should Be Equal for an explanation on how to override the default error message with msg and values.

should_start_with (str1, str2, msg=None, values=True)
Fails if the string str1 does not start with the string str2.

See Should Be Equal for an explanation on how to override the default error message with msg and values.

sleep (time_, reason=None)
Pauses the test executed for the given time.

time may be either a number or a time string. Time strings are in a format such as ‘1 day 2 hours 3 minutes 4 seconds 5milliseconds’ or ‘1d 2h 3m 4s 5ms’, and they are fully explained in an appendix of Robot Framework User Guide. Optional reason can be used to explain why sleeping is necessary. Both the time slept and the reason are logged.

Examples: | Sleep | 42 | | Sleep | 1.5 | | Sleep | 2 minutes 10 seconds | | Sleep | 10s | Wait for a reply |

variable_should_exist (name, msg=None)
Fails unless the given variable exists within the current scope.

The name of the variable can be given either as a normal variable name (e.g. ${NAME}) or in escaped format (e.g. ${NAME}). Notice that the former has some limitations explained in Set Suite Variable.

The default error message can be overridden with the msg argument.

See also Variable Should Not Exist and Keyword Should Exist.

variable_should_not_exist (name, msg=None)
Fails if the given variable exists within the current scope.

The name of the variable can be given either as a normal variable name (e.g. ${NAME}) or in escaped format (e.g. ${NAME}). Notice that the former has some limitations explained in Set Suite Variable.

The default error message can be overridden with the msg argument.

See also Variable Should Exist and Keyword Should Exist.

wait_until_keyword_succeeds (timeout, retry_interval, name, *args)
Waits until the specified keyword succeeds or the given timeout expires.

name and args define the keyword that is executed similarly as with Run Keyword. If the specified keyword does not succeed within timeout, this keyword fails. retry_interval is the time to wait before trying to run the keyword again after the previous run has failed.

Both timeout and retry_interval must be given in Robot Framework’s time format (e.g. ‘1 minute’, ‘2 min 3 s’, ‘4.5’).

Errors caused by invalid syntax, test or keyword timeouts, or fatal exceptions are not caught by this keyword.
Example: | Wait Until Keyword Succeeds | 2 min 15 sec | My keyword | arg1 | arg2 |

Running the same keyword multiple times inside this keyword can create lots of output and considerably increase the size of the generated output files. Starting from Robot Framework 2.7, it is possible to remove unnecessary keywords from the outputs using `--RemoveKeywords WUKS` command line option.

**robot.libraries.BuiltIn.register_run_keyword** *(library, keyword, args_to_process=None)*

Registers ‘run keyword’ so that its arguments can be handled correctly.

1. Why is this method needed

Keywords running other keywords internally (normally using `Run Keyword` or some variants of it in `BuiltIn`) must have the arguments meant to the internally executed keyword handled specially to prevent processing them twice. This is done ONLY for keywords registered using this method.

If the register keyword has same name as any keyword from Robot Framework standard libraries, it can be used without getting warnings. Normally there is a warning in such cases unless the keyword is used in long format (e.g. `MyLib.Keyword`).

Starting from Robot Framework 2.5.2, keywords executed by registered run keywords can be tested in dry-run mode they have ‘name’ argument which takes the name of the executed keyword.

2. How to use this method

*library* is the name of the library where the registered keyword is implemented.

*keyword* can be either a function or method implementing the keyword, or name of the implemented keyword as a string.

*args_to_process* is needed when *keyword* is given as a string, and it defines how many of the arguments to the registered keyword must be processed normally. When *keyword* is a method or function, this information is got directly from it so that varargs (those specified with syntax ‘*args’) are not processed but others are.

3. Examples

from robot.libraries.BuiltIn import BuiltIn, register_run_keyword

def my_run_keyword(name, *args): # do something return BuiltIn().run_keyword(name, *args)

# Either one of these works register_run_keyword(__name__, my_run_keyword) register_run_keyword(__name__, 'My Run Keyword', 1)

from robot.libraries.BuiltIn import BuiltIn, register_run_keyword

class MyLibrary:

def my_run_keyword_if(self, expression, name, *args): # do something return

    BuiltIn().run_keyword_if(expression, name, *args)

# Either one of these works register_run_keyword('MyLibrary', MyLibrary.my_run_keyword_if) register_run_keyword('MyLibrary', 'my_run_keyword_if', 2)

### 4.7.3 Collections Module

class robot.libraries.Collections.Collections


A test library providing keywords for handling lists and dictionaries.

*Collections* is Robot Framework’s standard library that provides a set of keywords for handling Python lists and dictionaries. This library has keywords, for example, for modifying and getting values from lists and dictionaries (e.g. *Append To List*, *Get From Dictionary*) and for verifying their contents (e.g. *Lists Should Be Equal*, *Dictionary Should Contain Value*).
Following keywords from the BuiltIn library can also be used with lists and dictionaries:

<table>
<thead>
<tr>
<th>Keyword Name</th>
<th>Applicable With</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create List</td>
<td>lists</td>
</tr>
<tr>
<td>Get Length</td>
<td>both</td>
</tr>
<tr>
<td>Length Should Be</td>
<td>both</td>
</tr>
<tr>
<td>Length Should Be</td>
<td>both</td>
</tr>
<tr>
<td>Should Be Empty</td>
<td>both</td>
</tr>
<tr>
<td>Should Be Empty</td>
<td>both</td>
</tr>
<tr>
<td>Should Contain</td>
<td>lists</td>
</tr>
<tr>
<td>Should Contain</td>
<td>lists</td>
</tr>
<tr>
<td>Should Contain X Times</td>
<td>lists</td>
</tr>
<tr>
<td>Should Not Contain</td>
<td>lists</td>
</tr>
<tr>
<td>Should Not Contain</td>
<td>lists</td>
</tr>
<tr>
<td>Should Not Contain</td>
<td>lists</td>
</tr>
<tr>
<td>Should Not Contain X Times</td>
<td>lists</td>
</tr>
<tr>
<td>Get Count</td>
<td>lists</td>
</tr>
</tbody>
</table>

All list keywords expect a scalar variable (e.g. `${list}$`) as an argument. It is, however, possible to use list variables (e.g. `@[list]`) as scalars simply by replacing `@` with `$`.

List keywords that do not alter the given list can also be used with tuples, and to some extent also with other iterables. `Convert To List` can be used to convert tuples and other iterables to lists.

List related keywords use variables in format `$\{Lx\}$` in their examples, which means a list with as many alphabetic characters as specified by `x`. For example `$\{L1\}$` means `['a']` and `$\{L3\}$` means `['a', 'b', 'c']`.

Dictionary keywords use similar `$\{Dx\}$` variables. For example `$\{D1\}$` means `{'a': 1}` and `$\{D3\}$` means `{'a': 1, 'b': 2, 'c': 3}`.

**ROBOT_LIBRARY_SCOPE** = ‘GLOBAL’

**ROBOT_LIBRARY_VERSION** = ‘2.7.7’

**append_to_list** (*list_, *values)

**Example:**

- `| Append To List | ${L1} | xxx | => - ${L1} = ['a', 'xxx']`
- `| Append To List | ${L2} | x | y | z | => - ${L2} = ['a', 'b', 'x', 'y', 'z']`

**combine_lists** (*lists)

**Example:**

- `| Combine List | ${L1} | ${L2} | => - ${x} = ['a', 'a', 'b'] - ${y} = ['a', 'a', 'b', 'a'] - ${L1} and ${L2} are not changed.`

**convert_to_list** (*item)

**Example:**

- `| Convert To List | ${L3} | ${b} | => - ${x} = ['a', 'b'] - ${y} = ['a', 'b', 'c'] - ${L3} is not changed.`

**create_dictionary** (*key_value_pairs)

**Examples:**

- `| Create Dictionary | name | value | => - ${x} = {'name': 'value'}`
- `| Create Dictionary | a | b | 2 | => - ${y} = {'a': '1', 'b': '2'} - ${z} = {'a': 1, 'b': 2}`
**dictionaries_should_be_equal** *(dict1, dict2, msg=None, values=True)*
Fails if the given dictionaries are not equal.

First the equality of dictionaries’ keys is checked and after that all the key value pairs. If there are differences between the values, those are listed in the error message.

See **Lists Should Be Equal** for an explanation of *msg*. The given dictionaries are never altered by this keyword.

**dictionary_should_contain_key** *(dictionary, key, msg=None)*
Fails if *key* is not found from *dictionary*.

See **List Should Contain Value** for an explanation of *msg*.

The given dictionary is never altered by this keyword.

**dictionary_should_contain_sub_dictionary** *(dict1, dict2, msg=None, values=True)*
Fails unless all items in *dict2* are found from *dict1*.

See **Lists Should Be Equal** for an explanation of *msg*. The given dictionaries are never altered by this keyword.

**dictionary_should_contain_value** *(dictionary, value, msg=None)*
Fails if *value* is not found from *dictionary*.

See **List Should Contain Value** for an explanation of *msg*.

The given dictionary is never altered by this keyword.

**dictionary_should_not_contain_key** *(dictionary, key, msg=None)*
Fails if *key* is found from *dictionary*.

See **List Should Contain Value** for an explanation of *msg*.

The given dictionary is never altered by this keyword.

**dictionary_should_not_contain_value** *(dictionary, value, msg=None)*
Fails if *value* is found from *dictionary*.

See **List Should Contain Value** for an explanation of *msg*.

The given dictionary is never altered by this keyword.

**get_dictionary_items** *(dictionary)*
Returns items of the given *dictionary*.

Items are returned sorted by keys. The given *dictionary* is not altered by this keyword.

Example: | $items = | Get Dictionary Items | $D3 | => - $items = ['a', 1, 'b', 2, 'c', 3]

**get_dictionary_keys** *(dictionary)*
Returns keys of the given *dictionary*.

Keys are returned in sorted order. The given *dictionary* is never altered by this keyword.

Example: | $keys = | Get Dictionary Keys | $D3 | => - $keys = ['a', 'b', 'c']

**get_dictionary_values** *(dictionary)*
Returns values of the given *dictionary*.

Values are returned sorted according to keys. The given dictionary is never altered by this keyword.

Example: | $values = | Get Dictionary Values | $D3 | => - $values = [1, 2, 3]

**get_from_dictionary** *(dictionary, key)*
Returns a value from the given *dictionary* based on the given *key*. 
If the given key cannot be found from the dictionary, this keyword fails.

The given dictionary is never altered by this keyword.

Example: | ${value} = | Get From Dictionary | ${D3} | b | => - ${value} = 2

get_from_list (list_, index)

Returns the value specified with an index from list.

The given list is never altered by this keyword.

Index ‘0’ means the first position, ‘1’ the second, and so on. Similarly, ‘-1’ is the last position, ‘-2’ the second last, and so on. Using an index that does not exist on the list causes an error. The index can be either an integer or a string that can be converted to an integer.

Examples (including Python equivalents in comments):
| $x = | Get From List | ${L5} | 0 | # L5[0] | |
| $y = | Get From List | ${L5} | -2 | # L5[-2] | => - $x = ‘a’ - $y = ‘d’ - ${L5} is not changed

get_index_from_list (list_, value, start=0, end=None)

Returns the index of the first occurrence of the value on the list.

The search can be narrowed to the selected sublist by the start and end indexes having the same semantics as in the Get Slice From List keyword. In case the value is not found, -1 is returned. The given list is never altered by this keyword.

Example: | ${x} = | Get Index From List | ${L5} | d | => - ${x} = 3 - ${L5} is not changed

get_slice_from_list (list_, start=0, end=None)

Returns a slice of the given list between start and end indexes.

The given list is never altered by this keyword.

If both start and end are given, a sublist containing values from start to end is returned. This is the same as ‘list[start:end]’ in Python. To get all items from the beginning, use 0 as the start value, and to get all items until the end, use ‘None’ as the end value. ‘None’ is also a default value, so in this case, it is enough to give only start. If only end is given, start gets the value 0.

Using start or end not found on the list is the same as using the largest (or smallest) available index.

Examples (incl. Python equivalents in comments):
| $x = | Get Slice From List | ${L5} | 2 | 4 | # L5[2:4]|
| $y = | Get Slice From List | ${L5} | 1 | 1 | 1 | # L5[1:None]|
| $z = | Get Slice From List | ${L5} | 1 | 1 | 1 | 2 | # L5[0:-2] | => - $x = [’c’, ’d’] - $y = [’b’, ’c’, ’d’, ’e’] - $z = [’a’, ’b’, ’c’] - ${L5} is not changed

insert_into_list (list_, index, value)

Inserts value into list to the position specified with index.

Index ‘0’ adds the value into the first position, ‘1’ to the second, and so on. Inserting from right works with negative indices so that ‘-1’ is the second last position, ‘-2’ third last, and so on. Use Append To List to add items to the end of the list.

If the absolute value of the index is greater than the length of the list, the value is added at the end (positive index) or the beginning (negative index). An index can be given either as an integer or a string that can be converted to an integer.

Example: | Insert Into List | ${L1} | 0 | xxx | | Insert Into List | ${L2} | -1 | xxx | => - ${L1} = [’xxx’, ’a’] - ${L2} = [’a’, ’xxx’, ’b’]

keep_in_dictionary (dictionary, *keys)

Keeps the given keys in the dictionary and removes all other.

If the given key cannot be found from the dictionary, it is ignored.

Example: | Keep In Dictionary | ${D5} | b | x | d | => - ${D5} = {’b’: 2, ’d’: 4}
list_should_contain_sub_list (list1, list2, msg=None, values=True)
Fails if not all of the elements in list2 are found in list1.

The order of values and the number of values are not taken into account.

See the use of msg and values from the Lists Should Be Equal keyword.

list_should_contain_value (list_, value, msg=None)
Fails if the value is not found from list.

If msg is not given, the default error message “[ a | b | c ] does not contain the value ‘x’” is shown in case of a failure. Otherwise, the given msg is used in case of a failure.

list_should_not_contain_duplicates (list_, msg=None)
Fails if any element in the list is found from it more than once.

The default error message lists all the elements that were found from the list multiple times, but it can be overridden by giving a custom msg. All multiple times found items and their counts are also logged.

This keyword works with all iterables that can be converted to a list. The original iterable is never altered.

list_should_not_contain_value (list_, value, msg=None)
Fails if the value is not found from list.

See List Should Contain Value for an explanation of msg.

lists_should_be_equal (list1, list2, msg=None, values=True, names=None)
Fails if given lists are unequal.

The keyword first verifies that the lists have equal lengths, and then it checks are all their values equal. Possible differences between the values are listed in the default error message like Index 4: ABC != Abc.

The error message can be configured using msg and values arguments: - If msg is not given, the default error message is used.
- If msg is given and values is either Boolean False or a string ‘False’ or ‘No Values’, the error message is simply msg.
- Otherwise the error message is msg + ‘new line’ + default.

Optional names argument (new in 2.6) can be used for naming the indices shown in the default error message. It can either be a list of names matching the indices in the lists or a dictionary where keys are indices that need to be named. It is not necessary to name all of the indices. When using a dictionary, keys can be either integers or strings that can be converted to integers.

Examples: | $\{names\} = | Create List | First Name | Family Name | Email | | Lists Should Be Equal | $\{people1\} | $\{people2\} | names=$\{names\} | Lists Should Be Equal | $\{people1\} | $\{people2\} | names=$\{names\} |

If the items in index 2 would differ in the above examples, the error message would contain a row like Index 2 (email): name@foo.com != name@bar.com.

log_dictionary (dictionary, level='INFO')
Logs the size and contents of the dictionary using given level.

Valid levels are TRACE, DEBUG, INFO (default), and WARN.

If you only want to log the size, use keyword Get Length from the BuiltIn library.

log_list (list_, level='INFO')
Logs the length and contents of the list using given level.

Valid levels are TRACE, DEBUG, INFO (default), and WARN.

If you only want to the length, use keyword Get Length from the BuiltIn library.
**remove_duplicates** *(list_)*  
Returns a list without duplicates based on the given list.

Creates and returns a new list that contains all items in the given list so that one item can appear only once. Order of the items in the new list is the same as in the original except for missing duplicates. Number of the removed duplicates is logged.

New in Robot Framework 2.7.5.

**remove_from_dictionary** *(dictionary, *keys)*  
Removes the given keys from the dictionary.

If the given key cannot be found from the dictionary, it is ignored.

Example: `| Remove From Dictionary | ${D3} | b | x | y | => - | ${D3} = \{'a': 1, 'c': 3\}`

**remove_from_list** *(list_, index)*  
Removes and returns the value specified with an index from list.

Index ‘0’ means the first position, ‘1’ the second and so on. Similarly, ‘-1’ is the last position, ‘-2’ the second last, and so on. Using an index that does not exist on the list causes an error. The index can be either an integer or a string that can be converted to an integer.

Example: `| Remove From List | ${L2} | 0 | => - | ${x} = 'a' - | ${L2} = \['b'\]`

**remove_values_from_list** *(list_, *values)*  
Removes all occurrences of given values from list.

It is not an error if a value does not exist in the list at all.

Example: `| Remove Values From List | ${L4} | a | c | e | f | => - | ${L4} = \['b', 'd'\]`

**reverse_list** *(list_)*  
Reverses the given list in place.

Note that the given list is changed and nothing is returned. Use Copy List first, if you need to keep also the original order.

Reverse List `| ${L3} | `  

`=> - | ${L3} = \['c', 'b', 'a'\]`

**set_list_value** *(list_, index, value)*  
Sets the value of list specified by index to the given value.

Index ‘0’ means the first position, ‘1’ the second and so on. Similarly, ‘-1’ is the last position, ‘-2’ second last, and so on. Using an index that does not exist on the list causes an error. The index can be either an integer or a string that can be converted to an integer.

Example: `| Set List Value | ${L3} | 1 | xxx | | Set List Value | ${L3} | -1 | yyy | => - | ${L3} = \['a', 'xxx', 'yyy'\]`

**set_to_dictionary** *(dictionary, *key_value_pairs)*  
Adds the given key_value_pairs to the dictionary.

Example: `| Set To Dictionary | ${D1} | key | value | => - | ${D1} = \{'a': 1, 'key': 'value'\}`

**sort_list** *(list_)*  
Sorts the given list in place.

The strings are sorted alphabetically and the numbers numerically.
Note that the given list is changed and nothing is returned. Use *Copy List* first, if you need to keep also the original order.

```python
$L = [2,1,'a','c','b'] | Sort List | $L | => - $L = [1, 2, 'a', 'b', 'c']
```

### 4.7.4 *DeprecatedBuiltIn* Module

Class `robot.libraries.DeprecatedBuiltIn.DerprecatedBuiltIn`

```python
ROBOT_LIBRARY_SCOPE = 'GLOBAL'

classmethod integer (item, base=None)
    Converts the given item to an integer number.

    If the given item is a string, it is by default expected to be an integer in base 10. Starting from Robot Framework 2.6 there are two ways to convert from other bases:

    1. Give base explicitly to the keyword as `base` argument.

    2) Prefix the given string with the base so that `0b` means binary (base 2), `0o` means octal (base 8), and `0x` means hex (base 16). The prefix is considered only when `base` argument is not given and may itself be prefixed with a plus or minus sign.

    The syntax is case-insensitive and possible spaces are ignored.

    Examples:

    ```
    | ${result} = | Convert To Integer | 100 | # Result is 100 |
    | ${result} = | Convert To Integer | FF AA | 16 | # Result is 65450 |
    | ${result} = | Convert To Integer | 10018 | # Result is 64 |
    | ${result} = | Convert To Integer | -100 | 2 | # Result is -4 |
    | ${result} = | Convert To Integer | 0b100 | # Result is 4 |
    | ${result} = | Convert To Integer | -0x100 | # Result is -256 |
    ```

    See also *Convert To Number*, *Convert To Binary*, *Convert To Octal* and *Convert To Hex*.

classmethod float (item, precision=None)
    Converts the given item to a floating point number.

    If the optional `precision` is positive or zero, the returned number is rounded to that number of decimal digits. Negative precision means that the number is rounded to the closest multiple of 10 to the power of the absolute precision. The support for precision was added in Robot Framework 2.6.

    Examples:

    ```
    | ${result} = | Convert To Number | 42.512 | # Result is 42.512 |
    | ${result} = | Convert To Number | 42.512 | 1 | # Result is 42.5 |
    | ${result} = | Convert To Number | 42.512 | 0 | # Result is 43.0 |
    | ${result} = | Convert To Number | 42.512 | -1 | # Result is 40.0 |
    ```

    Notice that machines generally cannot store floating point numbers accurately. This may cause surprises with these numbers in general and also when they are rounded. For more information see, for example, this floating point arithmetic tutorial: http://docs.python.org/tutorial/floatingpoint.html

    If you need an integer number, use *Convert To Integer* instead.

classmethod string (item)
    Converts the given item to a Unicode string.

    Uses `__unicode__` or `__str__` method with Python objects and `toString` with Java objects.

    Use *Encode String To Bytes* and *Decode Bytes To String* keywords in *String* library if you need to convert between Unicode and byte strings.

classmethod boolean (item)
    Converts the given item to Boolean true or false.

    Handles strings ‘True’ and ‘False’ (case-insensitive) as expected, otherwise returns item’s truth value using Python’s `bool` method. For more information about truth values, see http://docs.python.org/lib/truth.html.
classmethod **list** (*items*)

Returns a list containing given items.

The returned list can be assigned both to `${scalar}` and @{list} variables. The earlier can be used e.g. with Java keywords expecting an array as an argument.

Examples: | @{list} = | Create List | a | b | c | | ${scalar} = | Create List | a | b | c | | ${ints} = | Create List | ${1} | ${2} | ${3} |

```python

classmethod **equal** (first, second, msg=None, values=True)

Fails if the given objects are unequal.

- If `msg` is not given, the error message is ‘first != second’.
- If `msg` is given and `values` is either Boolean False or the string ‘False’ or ‘No Values’, the error message is simply `msg`.
- Otherwise the error message is ‘msg: first != second’.
```

```python

classmethod **equals** (first, second, msg=None, values=True)

Fails if the given objects are unequal.

- If `msg` is not given, the error message is ‘first != second’.
- If `msg` is given and `values` is either Boolean False or the string ‘False’ or ‘No Values’, the error message is simply `msg`.
- Otherwise the error message is ‘msg: first != second’.
```

```python

classmethod **fail_unless_equal** (first, second, msg=None, values=True)

Fails if the given objects are unequal.

- If `msg` is not given, the error message is ‘first != second’.
- If `msg` is given and `values` is either Boolean False or the string ‘False’ or ‘No Values’, the error message is simply `msg`.
- Otherwise the error message is ‘msg: first != second’.
```

```python

classmethod **not_equal** (first, second, msg=None, values=True)

Fails if the given objects are equal.

See Should Be Equal for an explanation on how to override the default error message with `msg` and `values`.

```python

classmethod **not_equals** (first, second, msg=None, values=True)

Fails if the given objects are equal.

See Should Be Equal for an explanation on how to override the default error message with `msg` and `values`.

```python

classmethod **fail_if_equal** (first, second, msg=None, values=True)

Fails if the given objects are equal.

See Should Be Equal for an explanation on how to override the default error message with `msg` and `values`.

```python

classmethod **is_true** (condition, msg=None)

Fails if the given condition is not true.

If `condition` is a string (e.g. ‘${rc} < 10’), it is evaluated as a Python expression using the built-in ‘eval’ function and the keyword status is decided based on the result. If a non-string item is given, the status is got directly from its truth value as explained at http://docs.python.org/lib/truth.html.

The default error message (‘<condition> should be true’) is not very informative, but it can be overridden with the `msg` argument.
Examples: | Should Be True | $\{rc\} < 10 | | Should Be True | ‘$\{status\}’ == ‘PASS’ | # Strings must be quoted | | Should Be True | $\{number\} | # Passes if $\{number\} is not zero | | Should Be True | $\{list\} | # Passes if $\{list\} is not empty |

classmethod **fail_unless**(condition, msg=None)
Fails if the given condition is not true.

If *condition* is a string (e.g. ‘$\{rc\} < 10’), it is evaluated as a Python expression using the built-in ‘eval’ function and the keyword status is decided based on the result. If a non-string item is given, the status is got directly from its truth value as explained at [http://docs.python.org/lib/truth.html](http://docs.python.org/lib/truth.html).

The default error message (‘<condition> should be true’) is not very informative, but it can be overridden with the *msg* argument.

Examples: | Should Be True | $\{rc\} < 10 | | Should Be True | ‘$\{status\}’ == ‘PASS’ | # Strings must be quoted | | Should Be True | $\{number\} | # Passes if $\{number\} is not zero | | Should Be True | $\{list\} | # Passes if $\{list\} is not empty |

classmethod **is_false**(condition, msg=None)
Fails if the given condition is true.

See *Should Be True* for details about how *condition* is evaluated and how *msg* can be used to override the default error message.

classmethod **fail_if**(condition, msg=None)
Fails if the given condition is true.

See *Should Be True* for details about how *condition* is evaluated and how *msg* can be used to override the default error message.

classmethod **fail_if_ints_equal**(first, second, msg=None, values=True, base=None)
Fails if objects are equal after converting them to integers.

See *Convert To Integer* for information how to convert integers from other bases than 10 using *base* argument or *0b/0o/0x* prefixes.

See *Should Be Equal* for an explanation on how to override the default error message with *msg* and *values*.

See *Should Be Equal As Integers* for some usage examples.

classmethod **ints_not_equal**(first, second, msg=None, values=True, base=None)
Fails if objects are equal after converting them to integers.

See *Convert To Integer* for information how to convert integers from other bases than 10 using *base* argument or *0b/0o/0x* prefixes.

See *Should Be Equal* for an explanation on how to override the default error message with *msg* and *values*.

See *Should Be Equal As Integers* for some usage examples.

classmethod **ints_equal**(first, second, msg=None, values=True, base=None)
Fails if objects are unequal after converting them to integers.

See *Convert To Integer* for information how to convert integers from other bases than 10 using *base* argument or *0b/0o/0x* prefixes.

See *Should Be Equal* for an explanation on how to override the default error message with *msg* and *values*.

Examples: | Should Be Equal As Integers | 42 | $\{42\} | Error message | | Should Be Equal As Integers | ABCD | abcd | base=16 | | Should Be Equal As Integers | 0b1011 | 11 |

classmethod **fail_unless_ints_equal**(first, second, msg=None, values=True, base=None)
Fails if objects are unequal after converting them to integers.
See Convert To Integer for information how to convert integers from other bases than 10 using base argument or 0b/0o/0x prefixes.

See Should Be Equal for an explanation on how to override the default error message with msg and values.

Examples: | Should Be Equal As Integers | 42 | {42} | Error message | | Should Be Equal As Integers | ABCD | abcd | base=16 | Should Be Equal As Integers | 0b1011 | 11 |

classmethod floats_not_equal (first, second, msg=None, values=True, precision=6)
Fails if objects are equal after converting them to real numbers.

The conversion is done with Convert To Number keyword using the given precision. The support for giving precision was added in Robot Framework 2.6, in earlier versions it was hard-coded to 6.

See Should Be Equal As Numbers for examples on how to use precision and why it does not always work as expected. See also Should Be Equal for an explanation on how to override the default error message with msg and values.

classmethod fail_if_floats_equal (first, second, msg=None, values=True, precision=6)
Fails if objects are equal after converting them to real numbers.

The conversion is done with Convert To Number keyword using the given precision. The support for giving precision was added in Robot Framework 2.6, in earlier versions it was hard-coded to 6.

See Should Be Equal As Numbers for examples on how to use precision and why it does not always work as expected. See also Should Be Equal for an explanation on how to override the default error message with msg and values.

classmethod floats_equal (first, second, msg=None, values=True, precision=6)
Fails if objects are unequal after converting them to real numbers.

The conversion is done with Convert To Number keyword using the given precision. The support for giving precision was added in Robot Framework 2.6, in earlier versions it was hard-coded to 6.

Examples: | Should Be Equal As Numbers | ${x} | 1.1 | | # Passes if ${x} is 1.1 | | Should Be Equal As Numbers | 1.123 | 1.1 | precision=1 | # Passes | | Should Be Equal As Numbers | 1.123 | 1.4 | precision=0 | # Passes | | Should Be Equal As Numbers | 112.3 | 75 | precision=-2 | # Passes |

As discussed in the documentation of Convert To Number, machines generally cannot store floating point numbers accurately. Because of this limitation, comparing floats for equality is problematic and a correct approach to use depends on the context. This keyword uses a very naive approach of rounding the numbers before comparing them, which is both prone to rounding errors and does not work very well if numbers are really big or small. For more information about comparing floats, and ideas on how to implement your own context specific comparison algorithm, see this great article: http://www.cygnsupport.com/papers/comparingfloats/comparingfloats.htm

See Should Not Be Equal As Numbers for a negative version of this keyword and Should Be Equal for an explanation on how to override the default error message with msg and values.

classmethod fail_unless_floats_equal (first, second, msg=None, values=True, precision=6)
Fails if objects are unequal after converting them to real numbers.

The conversion is done with Convert To Number keyword using the given precision. The support for giving precision was added in Robot Framework 2.6, in earlier versions it was hard-coded to 6.

Examples: | Should Be Equal As Numbers | ${x} | 1.1 | | # Passes if ${x} is 1.1 | | Should Be Equal As Numbers | 1.123 | 1.1 | precision=1 | # Passes | | Should Be Equal As Numbers | 1.123 | 1.4 | precision=0 | # Passes | | Should Be Equal As Numbers | 112.3 | 75 | precision=-2 | # Passes |

As discussed in the documentation of Convert To Number, machines generally cannot store floating point numbers accurately. Because of this limitation, comparing floats for equality is problematic and a correct approach to use depends on the context. This keyword uses a very naive approach of rounding the
numbers before comparing them, which is both prone to rounding errors and does not work very well if numbers are really big or small. For more information about comparing floats, and ideas on how to implement your own context specific comparison algorithm, see this great article: http://www.cygnus-software.com/papers/comparingfloats/comparingfloats.htm

See Should Not Be Equal As Numbers for a negative version of this keyword and Should Be Equal for an explanation on how to override the default error message with msg and values.

classmethod does_not_start (str1, str2, msg=None, values=True)
Fails if the string str1 starts with the string str2.
See Should Be Equal for an explanation on how to override the default error message with msg and values.

classmethod fail_if_starts (str1, str2, msg=None, values=True)
Fails if the string str1 starts with the string str2.
See Should Be Equal for an explanation on how to override the default error message with msg and values.

classmethod starts (str1, str2, msg=None, values=True)
Fails if the string str1 does not start with the string str2.
See Should Be Equal for an explanation on how to override the default error message with msg and values.

classmethod fail_unless_starts (str1, str2, msg=None, values=True)
Fails if the string str1 does not start with the string str2.
See Should Be Equal for an explanation on how to override the default error message with msg and values.

classmethod does_not_end (str1, str2, msg=None, values=True)
Fails if the string str1 ends with the string str2.
See Should Be Equal for an explanation on how to override the default error message with msg and values.

classmethod fail_if_ends (str1, str2, msg=None, values=True)
Fails if the string str1 ends with the string str2.
See Should Be Equal for an explanation on how to override the default error message with msg and values.

classmethod ends (str1, str2, msg=None, values=True)
Fails if the string str1 does not end with the string str2.
See Should Be Equal for an explanation on how to override the default error message with msg and values.

classmethod fail_unless_ends (str1, str2, msg=None, values=True)
Fails if the string str1 does not end with the string str2.
See Should Be Equal for an explanation on how to override the default error message with msg and values.

classmethod does_not_contain (item1, item2, msg=None, values=True)
Fails if item1 contains item2 one or more times.
Works with strings, lists, and anything that supports Python’s ‘in’ keyword. See Should Be Equal for an explanation on how to override the default error message with msg and values.
Examples: | Should Not Contain | ${output} | FAILED |

classmethod fail_if_contains (item1, item2, msg=None, values=True)
Fails if item1 contains item2 one or more times.
Works with strings, lists, and anything that supports Python’s ‘in’ keyword. See Should Be Equal for an explanation on how to override the default error message with msg and values.
Examples: | Should Not Contain | ${output} | FAILED |
**classmethod contains** *(item1, item2, msg=None, values=True)*
Fails if item1 does not contain item2 one or more times.

Works with strings, lists, and anything that supports Python’s ‘in’ keyword. See *Should Be Equal* for an explanation on how to override the default error message with msg and values.

Examples: |
| Should Contain | ${output} | PASS |
| Should Contain | ${some_list} | value |

**classmethod fail_unless_contains** *(item1, item2, msg=None, values=True)*
Fails if item1 does not contain item2 one or more times.

Works with strings, lists, and anything that supports Python’s ‘in’ keyword. See *Should Be Equal* for an explanation on how to override the default error message with msg and values.

Examples: |
| Should Contain | ${output} | PASS |
| Should Contain | ${some_list} | value |

**classmethod does_not_match** *(string, pattern, msg=None, values=True)*
Fails if the given string matches the given pattern.

Pattern matching is similar as matching files in a shell, and it is always case-sensitive. In the pattern ‘*’ matches to anything and ‘?’ matches to any single character.

See *Should Be Equal* for an explanation on how to override the default error message with msg and values.

**classmethod fail_if_matches** *(string, pattern, msg=None, values=True)*
Fails if the given string matches the given pattern.

Pattern matching is similar as matching files in a shell, and it is always case-sensitive. In the pattern ‘*’ matches to anything and ‘?’ matches to any single character.

See *Should Be Equal* for an explanation on how to override the default error message with msg and values.

**classmethod matches** *(string, pattern, msg=None, values=True)*
Fails unless the given string matches the given pattern.

Pattern matching is similar as matching files in a shell, and it is always case-sensitive. In the pattern ‘*’ matches to anything and ‘?’ matches to any single character.

See *Should Be Equal* for an explanation on how to override the default error message with msg and values.

**classmethod fail_unless_matches** *(string, pattern, msg=None, values=True)*
Fails unless the given string matches the given pattern.

Pattern matching is similar as matching files in a shell, and it is always case-sensitive. In the pattern ‘*’ matches to anything and ‘?’ matches to any single character.

See *Should Be Equal* for an explanation on how to override the default error message with msg and values.

**classmethod does_not_match_regexp** *(string, pattern, msg=None, values=True)*
Fails if string matches pattern as a regular expression.

See *Should Match Regexp* for more information about arguments.

**classmethod fail_if_regexp_matches** *(string, pattern, msg=None, values=True)*
Fails if string matches pattern as a regular expression.

See *Should Match Regexp* for more information about arguments.

**classmethod matches_regexp** *(string, pattern, msg=None, values=True)*
Fails if string does not match pattern as a regular expression.

Regular expression check is done using the Python ‘re’ module, which has a pattern syntax derived from Perl, and thus also very similar to the one in Java. See the following documents for more details about regular expressions in general and Python implementation in particular.
Things to note about the regexp syntax in Robot Framework test data:

1) Backslash is an escape character in the test data, and possible backslashes in the pattern must thus be escaped with another backslash (e.g. `\d\w+`).

2) Strings that may contain special characters, but should be handled as literal strings, can be escaped with the `Regexp Escape` keyword.

3) The given pattern does not need to match the whole string. For example, the pattern `ello` matches the string `Hello world!`. If a full match is needed, the `^` and `$` characters can be used to denote the beginning and end of the string, respectively. For example, `^ello$` only matches the exact string `ello`.

4) Possible flags altering how the expression is parsed (e.g. `re.IGNORECASE, re.MULTILINE`) can be set by prefixing the pattern with the `(?iLmsux)` group (e.g. `(?im)pattern`). The available flags are `IGNORECASE`: `i`, `MULTILINE`: `m`, `DOTALL`: `s`, `VERBOSE`: `x`, `UNICODE`: `u`, and `LOCALE`: `L`.

If this keyword passes, it returns the portion of the string that matched the pattern. Additionally, the possible captured groups are returned.

See the `Should Be Equal` keyword for an explanation on how to override the default error message with the `msg` and `values` arguments.

Examples:

| Should Match Regexp | ${output} | \d{6} | # Output contains six numbers |
| Should Match Regexp | ${output} | ^\d{6}$ | # Six numbers and nothing more |
| Should Match Regexp | Foo: 42 | (?i)foo: \d+ | => | Foo: 42 |
| Should Match Regexp | Bar: 43 | (Foo|Bar): (\d+) | => | Bar: 43 |

**classmethod fail_unless_regexp_matches**(string, pattern, msg=None, values=True)

Fails if `string` does not match `pattern` as a regular expression.

Regular expression check is done using the Python `re` module, which has a pattern syntax derived from Perl, and thus also very similar to the one in Java. See the following documents for more details about regular expressions in general and Python implementation in particular.

Things to note about the regexp syntax in Robot Framework test data:

1) Backslash is an escape character in the test data, and possible backslashes in the pattern must thus be escaped with another backslash (e.g. `\d\w+`).

2) Strings that may contain special characters, but should be handled as literal strings, can be escaped with the `Regexp Escape` keyword.

3) The given pattern does not need to match the whole string. For example, the pattern `ello` matches the string `Hello world!`. If a full match is needed, the `^` and `$` characters can be used to denote the beginning and end of the string, respectively. For example, `^ello$` only matches the exact string `ello`.

4) Possible flags altering how the expression is parsed (e.g. `re.IGNORECASE, re.MULTILINE`) can be set by prefixing the pattern with the `(?iLmsux)` group (e.g. `(?im)pattern`). The available flags are `IGNORECASE`: `i`, `MULTILINE`: `m`, `DOTALL`: `s`, `VERBOSE`: `x`, `UNICODE`: `u`, and `LOCALE`: `L`. 

276 Chapter 4. All packages
If this keyword passes, it returns the portion of the string that matched the pattern. Additionally, the possible captured groups are returned.

See the *Should Be Equal* keyword for an explanation on how to override the default error message with the *msg* and *values* arguments.

Examples:

```plaintext
| Should Match Regexp | ${output} | \d{6} | # Output contains six numbers |
| Should Match Regexp | ${output} | ^\d{6}\d| # Six numbers and nothing more |
| Should Match Regexp | Foo: 42\d+ | ${match} | ${group1} | ${group2} | # Should Match Regexp Foo: Bar: 43 (FoolBar): ('d+)| => | ${ret} = 'Fool: 42' | ${match} = 'Bar: 43' | ${group1} = 'Bar' | ${group2} = '43'
```

**classmethod `noop()`**

Does absolutely nothing.

**classmethod `set_(*values)`**

Returns the given values which can then be assigned to a variables.

This keyword is mainly used for setting scalar variables. Additionally it can be used for converting a scalar variable containing a list to a list variable or to multiple scalar variables. It is recommended to use *Create List* when creating new lists.

Examples:

```plaintext
| $hi | = | Set Variable | Hello, world! |
| $hi2 | = | Set Variable | I said: |
| $var1 | = | Set Variable | Hello | world |
| $var2 | = | Set Variable | ${list with some items} |
| @list | = | Set Variable | ${item1} |
| @item2 | = | Set Variable | ${list with 2 items} |
```

Variables created with this keyword are available only in the scope where they are created. See *Set Global Variable*, *Set Test Variable* and *Set Suite Variable* for information on how to set variables so that they are available also in a larger scope.

**classmethod `message(*messages)`**

Displays the given messages in the log file as keyword arguments.

This keyword does nothing with the arguments it receives, but as they are visible in the log, this keyword can be used to display simple messages. Given arguments are ignored so thoroughly that they can even contain non-existing variables. If you are interested about variable values, you can use the *Log* or *Log Many* keywords.

**classmethod `variable_exists(name, msg=None)`**

Fails unless the given variable exists within the current scope.

The name of the variable can be given either as a normal variable name (e.g. `${NAME}`) or in escaped format (e.g. `\${NAME}`). Notice that the former has some limitations explained in *Set Suite Variable*.

The default error message can be overridden with the *msg* argument.

See also *Variable Should Not Exist* and *Keyword Should Exist*.

**classmethod `fail_unless_variable_exists(name, msg=None)`**

Fails unless the given variable exists within the current scope.

The name of the variable can be given either as a normal variable name (e.g. `${NAME}`) or in escaped format (e.g. `\${NAME}`). Notice that the former has some limitations explained in *Set Suite Variable*.

The default error message can be overridden with the *msg* argument.

See also *Variable Should Not Exist* and *Keyword Should Exist*.

**classmethod `variable_does_not_exist(name, msg=None)`**

Fails if the given variable exists within the current scope.

The name of the variable can be given either as a normal variable name (e.g. `${NAME}`) or in escaped format (e.g. `\${NAME}`). Notice that the former has some limitations explained in *Set Suite Variable*.

The default error message can be overridden with the *msg* argument.
See also Variable Should Exist and Keyword Should Exist.

```python
classmethod fail_if_variable_exists (name, msg=None)
```
Fails if the given variable exists within the current scope.

The name of the variable can be given either as a normal variable name (e.g. `${NAME}`) or in escaped format (e.g. `$(NAME)`). Notice that the former has some limitations explained in Set Suite Variable.

The default error message can be overridden with the `msg` argument.

See also Variable Should Exist and Keyword Should Exist.

```python
classmethod delete_environment_variable (*names)
```
Deletes the specified environment variable.

Does nothing if the environment variable is not set.

Starting from Robot Framework 2.7, it is possible to remove multiple variables by passing them to this keyword as separate arguments.

```python
classmethod environment_variable_is_set (name, msg=None)
```
Fails if the specified environment variable is not set.

The default error message can be overridden with the `msg` argument.

```python
classmethod environment_variable_is_not_set (name, msg=None)
```
Fails if the specified environment variable is set.

The default error message can be overridden with the `msg` argument.

```python
classmethod fail_unless_exists (path, msg=None)
```
Fails unless the given path (file or directory) exists.

The path can be given as an exact path or as a glob pattern. The pattern matching syntax is explained in introduction. The default error message can be overridden with the `msg` argument.

```python
classmethod fail_if_exists (path, msg=None)
```
Fails if the given path (file or directory) exists.

The path can be given as an exact path or as a glob pattern. The pattern matching syntax is explained in introduction. The default error message can be overridden with the `msg` argument.

```python
classmethod fail_unless_file_exists (path, msg=None)
```
Fails unless the given path points to an existing file.

The path can be given as an exact path or as a glob pattern. The pattern matching syntax is explained in introduction. The default error message can be overridden with the `msg` argument.

```python
classmethod fail_if_file_exists (path, msg=None)
```
Fails if the given path points to an existing file.

4.7.5 DeprecatedOperatingSystem Module

```python
class robot.libraries.DeprecatedOperatingSystem.DeprecatedOperatingSystem
```

```python
ROBOT_LIBRARY_SCOPE = 'GLOBAL'
```

```python
classmethod delete_environment_variable (*names)
```
Deletes the specified environment variable.

Does nothing if the environment variable is not set.

Starting from Robot Framework 2.7, it is possible to remove multiple variables by passing them to this keyword as separate arguments.

```python
classmethod environment_variable_is_set (name, msg=None)
```
Fails if the specified environment variable is not set.

The default error message can be overridden with the `msg` argument.

```python
classmethod environment_variable_is_not_set (name, msg=None)
```
Fails if the specified environment variable is set.

The default error message can be overridden with the `msg` argument.

```python
classmethod fail_unless_exists (path, msg=None)
```
Fails unless the given path (file or directory) exists.

The path can be given as an exact path or as a glob pattern. The pattern matching syntax is explained in introduction. The default error message can be overridden with the `msg` argument.

```python
classmethod fail_if_exists (path, msg=None)
```
Fails if the given path (file or directory) exists.

The path can be given as an exact path or as a glob pattern. The pattern matching syntax is explained in introduction. The default error message can be overridden with the `msg` argument.

```python
classmethod fail_unless_file_exists (path, msg=None)
```
Fails unless the given path points to an existing file.

The path can be given as an exact path or as a glob pattern. The pattern matching syntax is explained in introduction. The default error message can be overridden with the `msg` argument.

```python
classmethod fail_if_file_exists (path, msg=None)
```
Fails if the given path points to an existing file.
The path can be given as an exact path or as a glob pattern. The pattern matching syntax is explained in *introduction*. The default error message can be overridden with the `msg` argument.

```python
classmethod fail_unless_dir_exists (path, msg=None)
```
Fails unless the given path points to an existing directory.

The path can be given as an exact path or as a glob pattern. The pattern matching syntax is explained in *introduction*. The default error message can be overridden with the `msg` argument.

```python
classmethod fail_if_dir_exists (path, msg=None)
```
Fails if the given path points to an existing file.

The path can be given as an exact path or as a glob pattern. The pattern matching syntax is explained in *introduction*. The default error message can be overridden with the `msg` argument.

```python
classmethod fail_unless_dir_empty (path, msg=None)
```
Fails unless the specified directory is empty.

The default error message can be overridden with the `msg` argument.

```python
classmethod fail_if_dir_empty (path, msg=None)
```
Fails if the specified directory is empty.

The default error message can be overridden with the `msg` argument.

```python
classmethod fail_unless_file_empty (path, msg=None)
```
Fails unless the specified file is empty.

The default error message can be overridden with the `msg` argument.

```python
classmethod fail_if_file_empty (path, msg=None)
```
Fails if the specified directory is empty.

The default error message can be overridden with the `msg` argument.

```python
classmethod empty_dir (path)
```
Deletes all the content (incl. subdirectories) from the given directory.

```python
classmethod remove_dir (path, recursive=False)
```
Removes the directory pointed to by the given `path`.

If the second argument `recursive` is set to any non-empty string, the directory is removed recursively. Otherwise removing fails if the directory is not empty.

If the directory pointed to by the `path` does not exist, the keyword passes, but it fails, if the `path` points to a file.

```python
classmethod copy_dir (source, destination)
```
Copies the source directory into the destination.

If the destination exists, the source is copied under it. Otherwise the destination directory and the possible missing intermediate directories are created.

```python
classmethod move_dir (source, destination)
```
Moves the source directory into a destination.

Uses *Copy Directory* keyword internally, and `source` and `destination` arguments have exactly same semantics as with that keyword.

```python
classmethod create_dir (path)
```
Creates the specified directory.

Also possible intermediate directories are created. Passes if the directory already exists, and fails if the path points to a regular file.
**classmethod list_dir** *(path, pattern=None, absolute=False)*

Returns and logs items in a directory, optionally filtered with `pattern`.

File and directory names are returned in case-sensitive alphabetical order, e.g. ['A Name', 'Second', 'a lower case name', 'one more']. Implicit directories '.' and '..' are not returned. The returned items are automatically logged.

By default, the file and directory names are returned relative to the given path (e.g. 'file.txt'). If you want them be returned in the absolute format (e.g. '/home/robot/file.txt'), set the `absolute` argument to any non-empty string.

If `pattern` is given, only items matching it are returned. The pattern matching syntax is explained in *introduction*, and in this case matching is case-sensitive.

Examples (using also other List Directory variants):  
| @{items} = | List Directory | ${TEMPDIR} |  
| @files = | List Files In Directory | /tmp | *.txt | absolute | | Count Files In Directory | ${CURDIR} | !!

**classmethod list_files_in_dir** *(path, pattern=None, absolute=False)*

A wrapper for List Directory that returns only files.

**classmethod list_dirs_in_dir** *(path, pattern=None, absolute=False)*

A wrapper for List Directory that returns only directories.

**classmethod count_items_in_dir** *(path, pattern=None)*

Returns and logs the number of all items in the given directory.

The argument `pattern` has the same semantics as in the List Directory keyword. The count is returned as an integer, so it must be checked e.g. with the built-in keyword *Should Be Equal As Integers*.

**classmethod count_files_in_dir** *(path, pattern=None)*

A wrapper for Count Items In Directory returning only file count.

**classmethod count_dirs_in_dir** *(path, pattern=None)*

A wrapper for Count Items In Directory returning only directory count.

### 4.7.6 Dialogs Module

A test library providing dialogs for interacting with users.

*Dialogs* is Robot Framework’s standard library that provides means for pausing the test execution and getting input from users. The dialogs are slightly different depending on are tests run on Python or Jython but they provide the same functionality.

The library has following two limitations: - It is not compatible with IronPython. - It cannot be used with timeouts on Python.

**robot.libraries.Dialogs.pause_execution** *(message='Test execution paused. Press OK to continue. ')*

Pauses test execution until user clicks Ok button.

`message` is the message shown in the dialog.

**robot.libraries.DIALOGS.execute_manual_step** *(message, default_error=' ')*

Pauses test execution until user sets the keyword status.

User can select ‘PASS’ or ‘FAIL’, and in the latter case an additional dialog is opened for defining the error message.

`message` is the instruction shown in the initial dialog and `default_error` is the default value shown in the possible error message dialog.
robot.libraries.DIALOGS.get_value_from_user(message, default_value='')
Pauses test execution and asks user to input a value.

message is the instruction shown in the dialog and default_value is the possible default value shown in the input field. Selecting ‘Cancel’ fails the keyword.

robot.libraries.DIALOGS.get_selection_from_user(message, *values)
Pauses test execution and asks user to select a value.

message is the instruction shown in the dialog and values are the options given to the user. Selecting ‘Cancel’ fails the keyword.

4.7.7 Easter Module

robot.libraries.EASTER.none_shall_pass(who)

4.7.8 OperatingSystem Module

class robot.libraries.OperatingSystem.OperatingSystem
A test library providing keywords for OS related tasks.

OperatingSystem is Robot Framework’s standard library that enables various operating system related tasks to be performed in the system where Robot Framework is running. It can, among other things, execute commands (e.g. Run), create and remove files and directories (e.g. Create File, Remove Directory), check whether files or directories exists or contain something (e.g. File Should Exist, Directory Should Be Empty) and manipulate environment variables (e.g. Set Environment Variable).

= Pattern matching =

Some keywords allow their arguments to be specified as _glob patterns_ where: | * | matches anything, even an empty string | ? | matches any single character | [chars] | matches any character inside square brackets (e.g. ‘[abc]’ matches either ‘a’, ‘b’ or ‘c’) | ![chars] | matches any character not inside square brackets |

Unless otherwise noted, matching is case-insensitive on case-insensitive operating systems such as Windows. Pattern matching is implemented using Python’s fnmatch module: http://docs.python.org/library/fnmatch.html

= Path separators =

All keywords expecting paths as arguments accept a forward slash (/) as a path separator regardless the operating system. Notice that this does not work when the path is part of an argument, like it often is with Run and Start Process keywords. In such cases the built-in variable ${/} can be used to keep the test data platform independent.

= Example =

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library</td>
<td>OperatingSystem</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>${PATH}</td>
<td>${CURDIR}/example.txt</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Action</th>
<th>Argument</th>
<th>Argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example</td>
<td>Create File</td>
<td>${PATH}</td>
<td>Some text</td>
</tr>
<tr>
<td></td>
<td>File Should Exist</td>
<td>${PATH}</td>
<td></td>
</tr>
</tbody>
</table>
```
| Copy File | ${PATH} | ${TEMPDIR}/stuff |
| ${output} = | Run | ${CURDIR}${/}script.py arg |
```

**ROBOT_LIBRARY_SCOPE** = ‘GLOBAL’

**ROBOT_LIBRARY_VERSION** = ‘2.7.7’

### run (command)
Runs the given command in the system and returns the output.

The execution status of the command is not checked by this keyword, and it must be done separately based on the returned output. If the execution return code is needed, either Run And Return RC or Run And Return RC And Output can be used.

The standard error stream is automatically redirected to the standard output stream by adding `2>&1` after the executed command. This automatic redirection is done only when the executed command does not contain additional output redirections. You can thus freely forward the standard error somewhere else, for example, like `my_command 2>stderr.txt`.

The returned output contains everything written into the standard output or error streams by the command (unless either of them is redirected explicitly). Many commands add an extra newline (`n`) after the output to make it easier to read in the console. To ease processing the returned output, this possible trailing newline is stripped by this keyword.

Examples:
```
| ${output} = | Run | ls -lhF /tmp | | Log | ${output} | | ${result} = | Run | ${CURDIR}${/}tester.py arg1 arg2 | | Should Not Contain | ${result} | FAIL | | ${stdout} = | Run | /opt/script.sh 2>/tmp/stderr.txt | | Should Be Equal | ${stdout} | TEST PASSED | | File Should Be Empty | /tmp/stderr.txt |
```

### run_and_return_rc (command)
Runs the given command in the system and returns the return code.

The return code (RC) is returned as a positive integer in range from 0 to 255 as returned by the executed command. On some operating systems (notable Windows) original return codes can be something else, but this keyword always maps them to the 0-255 range. Since the RC is an integer, it must be checked e.g. with the keyword Should Be Equal As Integers instead of Should Be Equal (both are built-in keywords).

Examples:
```
| ${rc} = | Run and Return RC | ${CURDIR}${/}script.py arg | | Should Be Equal As Integers | ${rc} | 0 | | ${rc} = | Run and Return RC | /path/to/example.rb arg1 arg2 | | Should Be True | 0 < ${rc} < 42 |
```

See Run and Run And Return RC And Output if you need to get the output of the executed command.

### run_and_return_rc_and_output (command)
Runs the given command in the system and returns the RC and output.

The return code (RC) is returned similarly as with Run And Return RC and the output similarly as with Run.

Examples:
```
| ${rc} | ${output} = | Run and Return RC And Output | ${CURDIR}${/}mytool | | Should Be Equal As Integers | ${rc} | 0 | | Should Not Contain | ${output} | FAIL | | ${rc} | ${stdout} = | Run and Return RC And Output | /opt/script.sh 2>/tmp/stderr.txt | | Should Be True | ${rc} > 42 | | Should Be Equal | ${stdout} | TEST PASSED | | File Should Be Empty | /tmp/stderr.txt |
```

### start_process (command, stdin=None, alias=None)
Starts the given command as a background process.

Starts the process in background and sets it as the active process. Read Process Output or Stop Process keywords affect this process unless Switch Process is used in between.
If the command needs input through the standard input stream, it can be defined with the `stdin` argument. It is not possible to give input to the command later. Possible command line arguments must be given as part of the command like `/tmp/script.sh arg1 arg2`.

Returns the index of this process. Indexing starts from 1, and indices can be used to switch between processes using `Switch Process` keyword. `Stop All Processes` can be used to reset indexing.

The optional `alias` is a name for this process that may be used with `Switch Process` instead of the returned index.

The standard error stream is redirected to the standard input stream automatically. This is done for the same reasons as with `Run` keyword, but redirecting is done when the process is started and not by adding `2>&1` to the command.

Example: |
---
Start Process | /path/longlasting.sh | Do Something | Expected text |
---
| $output | 

**switch_process (index_or_alias)**

Switches the active process to the specified process.

New active process can be specified either using an index or an alias. Indices are return values from `Start Process` and aliases can be given to that keyword.

Example: |
---
Start Process | /path/script.sh arg | alias=1st process | 2nd |
---
| Start Process | /path/script2.sh | Switch Process | 1st process | 2nd |
---
| Read Process Output | Switch Process | $output | |

**read_process_output ()**

Waits for a process to finish and returns its output.

This keyword waits for a process started with `Start Process` to end and then returns all output it has produced. The returned output contains everything the process has written into the standard output and error streams.

There is no need to use `Stop Process` after using this keyword. Trying to read from an already stopped process fails.

Note that although the process is finished, it still stays as the active process. Use `Switch Process` to switch the active process or `Stop All Processes` to reset the list of started processes.

**stop_process ()**

Closes the standard output stream of the process.

This keyword does not actually stop the process nor even wait for it to terminate. Only thing it does is closing the standard output stream of the process. Depending on the process that may terminate it but that is not guaranteed. Use `Read Process Output` instead if you need to wait for the process to complete.

This keyword operates the active process similarly as `Read Process Output`. Stopping an already stopped process is not an error.

**stop_all_processes ()**

Closes the standard output of all the processes and resets the process list.

Exactly like `Stop Process`, this keyword does not actually stop processes nor even wait for them to terminate.

This keyword resets the indexing that `Start Process` uses. All aliases are also deleted. It does not matter have some of the processes already been closed or not.

**get_file (path, encoding=’UTF-8’)**

Returns the contents of a specified file.
This keyword reads the specified file and returns the contents. Line breaks in content are converted to platform independent form. See also Get Binary File.

`encoding` defines the encoding of the file. By default the value is 'UTF-8', which means that UTF-8 and ASCII-encoded files are read correctly.

get_binary_file(path)

Returns the contents of a specified file.

This keyword reads the specified file and returns the contents as is. See also Get File.

New in Robot Framework 2.5.5.

grep_file(path, pattern, encoding='UTF-8')

Returns the lines of the specified file that match the pattern.

This keyword reads a file from the file system using the defined path and encoding similarly as Get File. A difference is that only the lines that match the given pattern are returned. Lines are returned as a single string catenated back together with newlines and the number of matched lines is automatically logged. Possible trailing newline is never returned.

A line matches if it contains the pattern anywhere in it and it does not need to match the pattern fully. The pattern matching syntax is explained in introduction, and in this case matching is case-sensitive.

Examples:

| ${errors} = | Grep File | /var/log/myapp.log | ERROR |
| ${ret} = | Grep File | ${CUR-DIR}/file.txt | [Ww]ildc??d ex*ple |

If more complex pattern matching is needed, it is possible to use Get File in combination with String library keywords like Get Lines Matching Regexp.

log_file(path, encoding='UTF-8')

Wrapper for Get File that also logs the returned file.

The file is logged with the INFO level. If you want something else, just use Get File and the built-in keyword Log with the desired level.

should_exist(path, msg=None)

Fails unless the given path (file or directory) exists.

The path can be given as an exact path or as a glob pattern. The pattern matching syntax is explained in introduction. The default error message can be overridden with the msg argument.

should_not_exist(path, msg=None)

Fails if the given path (file or directory) exists.

The path can be given as an exact path or as a glob pattern. The pattern matching syntax is explained in introduction. The default error message can be overridden with the msg argument.

file_should_exist(path, msg=None)

Fails unless the given path points to an existing file.

The path can be given as an exact path or as a glob pattern. The pattern matching syntax is explained in introduction. The default error message can be overridden with the msg argument.

file_should_not_exist(path, msg=None)

Fails if the given path points to an existing file.

The path can be given as an exact path or as a glob pattern. The pattern matching syntax is explained in introduction. The default error message can be overridden with the msg argument.

directory_should_exist(path, msg=None)

Fails unless the given path points to an existing directory.
The path can be given as an exact path or as a glob pattern. The pattern matching syntax is explained in
introduction. The default error message can be overridden with the msg argument.

directory_should_not_exist(path, msg=None)
Fails if the given path points to an existing file.

The path can be given as an exact path or as a glob pattern. The pattern matching syntax is explained in
introduction. The default error message can be overridden with the msg argument.

wait_until_removed(path, timeout='1 minute')
Waits until the given file or directory is removed.

The path can be given as an exact path or as a glob pattern. The pattern matching syntax is explained in
introduction. If the path is a pattern, the keyword waits until all matching items are removed.

The optional timeout can be used to control the maximum time of waiting. The timeout is given as a
timeout string, e.g. in a format ‘15 seconds’, ‘1min 10s’ or just ‘10’. The time string format is described

If the timeout is negative, the keyword is never timed-out. The keyword returns immediately, if the path
does not exist in the first place.

wait_until_created(path, timeout='1 minute')
Waits until the given file or directory is created.

The path can be given as an exact path or as a glob pattern. The pattern matching syntax is explained in
introduction. If the path is a pattern, the keyword returns when an item matching it is created.

The optional timeout can be used to control the maximum time of waiting. The timeout is given as a
timeout string, e.g. in a format ‘15 seconds’, ‘1min 10s’ or just ‘10’. The time string format is described

If the timeout is negative, the keyword is never timed-out. The keyword returns immediately, if the path
already exists.

directory_should_be_empty(path, msg=None)
Fails unless the specified directory is empty.

The default error message can be overridden with the msg argument.

directory_should_not_be_empty(path, msg=None)
Fails if the specified directory is empty.

The default error message can be overridden with the msg argument.

file_should_be_empty(path, msg=None)
Fails unless the specified file is empty.

The default error message can be overridden with the msg argument.

file_should_not_be_empty(path, msg=None)
Fails if the specified directory is empty.

The default error message can be overridden with the msg argument.

create_file(path, content='', encoding='UTF-8')
Creates a file with the given content and encoding.

If the directory where to create file does not exist it, and possible intermediate missing directories, are
created.

Use Append To File if you want to append to an existing file, and use File Should Not Exist if you want to
avoid overwriting existing files.
append_to_file(path, content, encoding='UTF-8')
Appends the given content to the specified file.

If the file does not exist, this keyword works exactly the same way as Create File With Encoding.

remove_file(path)
Removes a file with the given path.

Passes if the file does not exist, but fails if the path does not point to a regular file (e.g. it points to a directory).

The path can be given as an exact path or as a glob pattern. The pattern matching syntax is explained in introduction. If the path is a pattern, all files matching it are removed.

remove_files(*paths)
Uses Remove File to remove multiple files one-by-one.

Example: | Remove Files | \${TEMPDIR}{\}/foo.txt | \${TEMPDIR}{\}/bar.txt | \${TEMPDIR}{\}/zap.txt |

empty_directory(path)
Deletes all the content (incl. subdirectories) from the given directory.

create_directory(path)
Creates the specified directory.

Also possible intermediate directories are created. Passes if the directory already exists, and fails if the path points to a regular file.

remove_directory(path, recursive=False)
Removes the directory pointed to by the given path.

If the second argument recursive is set to any non-empty string, the directory is removed recursively. Otherwise removing fails if the directory is not empty.

If the directory pointed to by the path does not exist, the keyword passes, but it fails, if the path points to a file.

copy_file(source, destination)
Copies the source file into a new destination.

1) If the destination is an existing file, the source file is copied over it.

2) If the destination is an existing directory, the source file is copied into it. A possible file with the same name is overwritten.

3) If the destination does not exist and it ends with a path separator ('/' or '\'), it is considered a directory. That directory is created and a source file copied into it. Possible missing intermediate directories are also created.

4) If the destination does not exist and it does not end with a path separator, it is considered a file. If the path to the file does not exist, it is created.

move_file(source, destination)
Moves the source file into a new destination.

Uses Copy File keyword internally, and source and destination arguments have exactly same semantics as with that keyword.

copy_directory(source, destination)
Copies the source directory into the destination.

If the destination exists, the source is copied under it. Otherwise the destination directory and the possible missing intermediate directories are created.
move_directory (source, destination)
Moves the source directory into a destination.

Uses Copy Directory keyword internally, and source and destination arguments have exactly same semantics as with that keyword.

get_environment_variable (name, default=None)
Returns the value of an environment variable with the given name.

If no such environment variable is set, returns the default value, if given. Otherwise fails the test case.

Starting from Robot Framework 2.7, returned variables are automatically decoded to Unicode using the system encoding.

Note that you can also access environment variables directly using the variable syntax %[ENV_VAR_NAME].

set_environment_variable (name, value)
Sets an environment variable to a specified value.

Values are converted to strings automatically. Starting from Robot Framework 2.7, set variables are automatically encoded using the system encoding.

remove_environment_variable (*names)
deletes the specified environment variable.

Does nothing if the environment variable is not set.

Starting from Robot Framework 2.7, it is possible to remove multiple variables by passing them to this keyword as separate arguments.

environment_variable_should_be_set (name, msg=None)
Fails if the specified environment variable is not set.

The default error message can be overridden with the msg argument.

environment_variable_should_not_be_set (name, msg=None)
Fails if the specified environment variable is set.

The default error message can be overridden with the msg argument.

get_environment_variables ()
Returns currently available environment variables as a dictionary.

Both keys and values are decoded to Unicode using the system encoding. Altering the returned dictionary has no effect on the actual environment variables.

New in Robot Framework 2.7.

log_environment_variables (level='INFO')
Logs all environment variables using the given log level.

Environment variables are also returned the same way as with Get Environment Variables keyword.

New in Robot Framework 2.7.

join_path (base, *parts)
Joins the given path part(s) to the given base path.

The path separator (‘/’ or ‘\’) is inserted when needed and the possible absolute paths handled as expected. The resulted path is also normalized.

Examples: | $path = | Join Path | my | path | $p2 = | Join Path | my | path | $p3 = | Join Path | my | path | my | file.txt | $p4 = | Join Path | my | path | $p5 = | Join Path | my | path | .. | path2 |
join_paths (base, *paths)
Joins given paths with base and returns resulted paths.
See Join Path for more information.
Examples:
- @ {p1} = ['base/example', 'base/other']  - @ {p2} = ['/example', '/my/base/other']  - @ {p3} = ['my/base/example/path', 'my/base/other', 'my/base/one/more']

normalize_path (path)
Normalizes the given path.
Examples:
- ${path} = 'abc'  - ${p2} = 'abc'  - ${p3} = 'def'  - ${p4} = 'abc/def'  - ${p5} = 'abc/def'

split_path (path)
Splits the given path from the last path separator ('/' or 'a').
The given path is first normalized (e.g. a possible trailing path separator is removed, special directories '...' and '.' removed). The parts that are split are returned as separate components.
Examples:
- ${path1} & ${dir} = 'abc/def'  - ${path2} & ${file} = 'abc/def/ghi.txt'  - ${path3} & ${d2} = 'abc/../def/ghi'  =>  - ${path1} = 'abc' & ${dir} = 'def'  - ${path2} = 'abc/def' & ${file} = 'ghi.txt'  - ${path3} = 'def' & ${d2} = 'ghi'

split_extension (path)
Splits the extension from the given path.
The given path is first normalized (e.g. possible trailing path separators removed, special directories '...' and '.' removed). The base path and extension are returned as separate components so that the dot used as an extension separator is removed. If the path contains no extension, an empty string is returned for it. Possible leading and trailing dots in the file name are never considered to be extension separators.
Examples:
- ${path} & ${ext} = 'file.extension'  - ${p2} & ${e2} = 'path/file.ext'  - ${p3} & ${e3} = 'p1/../p2/file.ext'  - ${e5} = 'path/file.ext'  - ${p4} & ${e4} = 'path/file.ext'  - ${p6} & ${e6} = 'path/file.ext'  =>  - ${path} = 'file' & ${ext} = 'extension'  - ${p2} = 'path/file' & ${e2} = 'ext'  - ${p3} = 'path/file' & ${e3} = ''  - ${p4} = 'p2/file' & ${e4} = 'ext'  - ${p5} = 'path/file' & ${e5} = 'ext'  - ${p6} = 'path/file' & ${e6} = '''

get_modified_time (path, format='timestamp')
Returns the last modification time of a file or directory.
How time is returned is determined based on the given format string as follows. Note that all checks are case-insensitive. Returned time is also automatically logged.

1. If format contains the word 'epoch', the time is returned in seconds after the UNIX epoch. The return value is always an integer.

2. If format contains any of the words 'year', 'month', 'day', 'hour', 'min' or 'sec', only the selected parts are returned. The order of the returned parts is always the one in the previous sentence and the order of the words in format is not significant. The parts are returned as zero-padded strings (e.g. May -> '05').

3. Otherwise, and by default, the time is returned as a timestamp string in the format '2006-02-24 15:08:31'.
Examples (when the modified time of the ${CURDIR} is 2006-03-29 15:06:21): | ${time} = | Get Modified Time | ${CURDIR} | | ${secs} = | Get Modified Time | ${CURDIR} | epoch | | ${year} = | Get Modified Time | ${CURDIR} | return year | | ${y} | ${d} = | Get Modified Time | ${CURDIR} | year,day | | @{time} = | Get Modified Time | ${CURDIR} | year,month,day,hour,min,sec | => - ${time} = '2006-03-29 15:06:21' - ${secs} = 1143637581 - ${year} = '2006' - ${y} = '2006' & ${d} = '29' - @{time} = ['2006', '03', '29', '15', '06', '21']

set_modified_time (path, mtime)
Sets the file modification and access times.

Changes the modification and access times of the given file to the value determined by mtime. The time can be given in different formats described below. Note that all checks involving strings are case-insensitive.

1. If mtime is a number, or a string that can be converted to a number, it is interpreted as seconds since the UNIX epoch (1970-01-01 00:00:00 UTC). This documentation was originally written about 1177654467 seconds after the epoch.

2. If mtime is a timestamp, that time will be used. Valid timestamp formats are ‘YYYY-MM-DD hh:mm:ss’ and ‘YYYYMMDD hhmmss’.

3. If mtime is equal to ‘NOW’, the current local time is used. This time is got using Python’s ‘time.time()’ function.

4. If mtime is equal to ‘UTC’, the current time in [http://en.wikipedia.org/wiki/Coordinated_Universal_Time|UTC] is used. This time is got using ‘time.time() + time.altzone’ in Python.

5. If mtime is in the format like ‘NOW - 1 day’ or ‘UTC + 1 hour 30 min’, the current local/UTC time plus/minus the time specified with the time string is used. The time string format is described in an appendix of Robot Framework User Guide.

Examples: | Set Modified Time | /path/file | 1177654467 | # Time given as epoch seconds | | Set Modified Time | /path/file | 2007-04-27 9:14:27 | # Time given as a timestamp | | Set Modified Time | /path/file | NOW | # The local time of execution | | Set Modified Time | /path/file | NOW - 1 day | # 1 day subtracted from the local time | | Set Modified Time | /path/file | UTC + 1h 2min 3s | # 1h 2min 3s added to the UTC time |

Support for UTC time is a new feature in Robot Framework 2.7.5.

get_file_size (path)
Returns and logs file size as an integer in bytes

list_directory (path, pattern=None, absolute=False)
Returns and logs items in a directory, optionally filtered with pattern.

File and directory names are returned in case-sensitive alphabetical order, e.g. ['A Name', 'Second', 'a lower case name', 'one more']. Implicit directories '.' and '..' are not returned. The returned items are automatically logged.

By default, the file and directory names are returned relative to the given path (e.g. ‘file.txt’). If you want them be returned in the absolute format (e.g. ‘/home/robot/file.txt’), set the absolute argument to any non-empty string.

If pattern is given, only items matching it are returned. The pattern matching syntax is explained in introduction, and in this case matching is case-sensitive.

Examples (using also other List Directory variants): | @ {items} = | List Directory | ${TEMPDIR} | | @ {files} = | List Files In Directory | /tmp | * . txt | absolute | | ${count} = | Count Files In Directory | ${CURDIR} | ???
list_directories_in_directory (path, pattern=None, absolute=False)
   A wrapper for List Directory that returns only directories.

count_items_in_directory (path, pattern=None)
   Returns and logs the number of all items in the given directory.
   The argument pattern has the same semantics as in the List Directory keyword. The count is returned as an integer, so it must be checked e.g. with the built-in keyword Should Be Equal As Integers.

count_files_in_directory (path, pattern=None)
   A wrapper for Count Items In Directory returning only file count.

count_directories_in_directory (path, pattern=None)
   A wrapper for Count Items In Directory returning only directory count.

touch (path)
   Emulates the UNIX touch command.
   Creates a file, if it does not exist. Otherwise changes its access and modification times to the current time.
   Fails if used with the directories or the parent directory of the given file does not exist.

4.7.9 Remote Module

   ROBOT_LIBRARY_SCOPE = 'TEST SUITE'
   get_keyword_names (attempts=5)
   get_keyword_arguments (name)
   get_keyword_documentation (name)
   run_keyword (name, args)

class robot.libraries.Remote.RemoteResult (result)

class robot.libraries.Remote.XmlRpcRemoteClient (uri)
   get_keyword_names ()
   get_keyword_arguments (name)
   get_keyword_documentation (name)
   run_keyword (name, args)

4.7.10 Reserved Module

class robot.libraries.Reserved.Reserved
   ROBOT_LIBRARY_SCOPE = 'GLOBAL'
   get_keyword_names ()
   run_keyword (name, args)
4.7.11 Screenshot Module

class robot.libraries.Screenshot.Screenshot (screenshot_directory=None)

Bases: object

Test library for taking screenshots on the machine where tests are run.

Notice that successfully taking screenshots requires tests to be run with a physical or virtual display.

Using with Python

With Python you need to have one of the following modules installed to be able to use this library. The first module that is found will be used.

• wxPython :: http://wxpython.org :: Required also by RIDE so many Robot Framework users already have this module installed.
• PyGTK :: http://pygtk.org :: This module is available by default on most Linux distributions.
• Python Imaging Library (PIL) :: http://www.pythonware.com/products/pil :: This module can take screenshots only on Windows.

Python support was added in Robot Framework 2.5.5.

Using with Jython and IronPython

With Jython and IronPython this library uses APIs provided by JVM and .NET platforms, respectively. These APIs are always available and thus no external modules are needed.

IronPython support was added in Robot Framework 2.7.5.

Where screenshots are saved

By default screenshots are saved into the same directory where the Robot Framework log file is written. If no log is created, screenshots are saved into the directory where the XML output file is written.

It is possible to specify a custom location for screenshots using

screenshot_directory argument in importing and Set Screenshot Directory keyword during execution. It is also possible to save screenshots using an absolute path.

Note that prior to Robot Framework 2.5.5 the default screenshot location was system’s temporary directory.

Changes in Robot Framework 2.5.5 and Robot Framework 2.6

This library was heavily enhanced in Robot Framework 2.5.5 release. The changes are listed below and explained more thoroughly in affected places.

• The support for using this library on Python (see above) was added.
• The default location where screenshots are saved was changed (see above).
• New Take Screenshot and Take Screenshot Without Embedding keywords were added. These keywords should be used for taking screenshots in the future. Other screenshot taking keywords will be deprecated and removed later.
• log_file_directory argument was deprecated everywhere it was used.

In Robot Framework 2.6, following additional changes were made:

• log_file_directory argument was removed altogether.
• Set Screenshot Directories keyword was removed.
• *Save Screenshot, Save Screenshot To and Log Screenshot* keywords were deprecated. They will be removed in Robot Framework 2.8.

Configure where screenshots are saved.

If `screenshot_directory` is not given, screenshots are saved into same directory as the log file. The directory can also be set using `Set Screenshot Directory` keyword.

Examples (use only one of these):

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
<th>Value</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library</td>
<td>Screenshot</td>
<td>$\text{TEMPDIR}$</td>
<td># System temp (this was default prior to 2.5.5)</td>
</tr>
</tbody>
</table>

```text
ROBOT_LIBRARY_SCOPE = 'TEST SUITE'
ROBOT_LIBRARY_VERSION = '2.7.7'
```

**set_screenshot_directory** (*path*)

Sets the directory where screenshots are saved.

It is possible to use `/` as a path separator in all operating systems. Path to the old directory is returned.

The directory can also be set in `importing`.

**save_screenshot_to** (*path*)

*DEPRECATED* Use `Take Screenshot` or `Take Screenshot Without Embedding` instead.

**save_screenshot** (**basename**=`'screenshot'`, **directory**=None)

*DEPRECATED* Use `Take Screenshot` or `Take Screenshot Without Embedding` instead.

**log_screenshot** (**basename**=`'screenshot'`, **directory**=None, **width**=`'100\%'`)

*DEPRECATED* Use `Take Screenshot` or `Take Screenshot Without Embedding` instead.

**take_screenshot** (**name**=`'screenshot'`, **width**=`'800px'`)

Takes a screenshot in JPEG format and embeds it into the log file.

Name of the file where the screenshot is stored is derived from the given `name`. If the `name` ends with extension `.jpg` or `.jpeg`, the screenshot will be stored with that exact name. Otherwise a unique name is created by adding an underscore, a running index and an extension to the `name`.

The name will be interpreted to be relative to the directory where the log file is written. It is also possible to use absolute paths. Using `/` as a path separator works in all operating systems.

`width` specifies the size of the screenshot in the log file.

Examples: (LOGDIR is determined automatically by the library) | Take Screenshot | | | # LOGDIR/screenshot_1.jpg (index automatically incremented) | | Take Screenshot | mypic | | # LOGDIR/mypic_1.jpg (index automatically incremented) | | Take Screenshot | $\text{TEMPDIR}/mypic | | # /tmp/mypic_1.jpg (index automatically incremented) | | Take Screenshot | pic.jpg | | # LOGDIR/pic.jpg (always uses this file) | | Take Screenshot | images/login.jpg | 80% | # Specify both name and width. | | Take Screenshot | width=550px | | # Specify only width. | |

The path where the screenshot is saved is returned.

**take_screenshot_without_embedding** (**name**=`'screenshot'`)

Takes a screenshot and links it from the log file.

This keyword is otherwise identical to `Take Screenshot` but the saved screenshot is not embedded into the log file. The screenshot is linked so it is nevertheless easily available.
class robot.libraries.Screenshot.ScreenshotTaker (module_name=None)
    Bases: object

4.7.12 String Module

class robot.libraries.String.String
    A test library for string manipulation and verification.

    String is Robot Framework’s standard library for manipulating strings (e.g. Replace String Using Regexp, Split To Lines) and verifying their contents (e.g. Should Be String).

    Following keywords from BuiltIn library can also be used with strings:
    • Catenate
    • Get Length
    • Length Should Be
    • Should (Not) Be Empty
    • Should (Not) Be Equal (As Strings/Integers/Numbers)
    • Should (Not) Match (Regexp)
    • Should (Not) Contain
    • Should (Not) Start With
    • Should (Not) End With
    • Convert To String

    ROBOT_LIBRARY_SCOPE = 'GLOBAL'
    ROBOT_LIBRARY_VERSION = '2.7.7'

    encode_string_to_bytes (string, encoding, errors='strict')
    Encodes the given Unicode string to bytes using the given encoding.

    errors argument controls what to do if encoding some characters fails. All values accepted by encode method in Python are valid, but in practice the following values are most useful:
    • strict: fail if characters cannot be encoded (default)
    • ignore: ignore characters that cannot be encoded
    • replace: replace characters that cannot be encoded with a replacement character

    Examples: | ${bytes} = | Encode String To Bytes | ${string} | UTF-8 | | ${bytes} = | Encode String To Bytes | ${string} | ASCII | errors=ignore |

    Use Decode Bytes To String if you need to convert byte strings to Unicode strings, and Convert To String in BuiltIn if you need to convert arbitrary objects to Unicode strings.

    New in Robot Framework 2.7.7.

    decode_bytes_to_string (bytes, encoding, errors='strict')
    Decodes the given bytes to a Unicode string using the given encoding.

    errors argument controls what to do if decoding some bytes fails. All values accepted by decode method in Python are valid, but in practice the following values are most useful:
    • strict: fail if characters cannot be decoded (default)
    • ignore: ignore characters that cannot be decoded
•replace: replace characters that cannot be decoded with a replacement character

Examples: | ${string} = | Decode Bytes To String | ${bytes} | UTF-8 | | ${string} = | Decode Bytes To String | ${bytes} | ASCII | errors=ignore |

Use Encode String To Bytes if you need to convert Unicode strings to byte strings, and Convert To String in BuiltIn if you need to convert arbitrary objects to Unicode strings.

New in Robot Framework 2.7.7.

get_line_count(string)
Returns and logs the number of lines in the given string.

split_to_lines(string, start=0, end=None)
Converts the string into a list of lines.

It is possible to get only a selection of lines from start to end so that start index is inclusive and end is exclusive. Line numbering starts from 0, and it is possible to use negative indices to refer to lines from the end.

Lines are returned without the newlines. The number of returned lines is automatically logged.

Examples: |
| @{lines} = | Split To Lines | ${manylines} | | | | @{ignore first} = | Split To Lines | ${many-lines} | 1 | | | @{ignore last} = | Split To Lines | ${many-lines} | -1 | | | @{5th to 10th} = | Split To Lines | ${manylines} | 4 | 10 | | @{first two} = | Split To Lines | ${manylines} | 1 | | | @{last two} = | Split To Lines | ${manylines} | -2 | |

Use Get Line if you only need to get a single line.

get_line(string, line_number)
Returns the specified line from the given string.

Line numbering starts from 0 and it is possible to use negative indices to refer to lines from the end. The line is returned without the newline character.

Examples: |
| ${first} = | Get Line | ${string} | 0 | | | ${2nd last} = | Get Line | ${string} | -2 |

get_lines_containing_string(string, pattern, case_insensitive=False)
Returns lines of the given string that contain the pattern.

The pattern is always considered to be a normal string and a line matches if the pattern is found anywhere in it. By default the match is case-sensitive, but setting case_insensitive to any value makes it case-insensitive.

Lines are returned as one string concatenated back together with newlines. Possible trailing newline is never returned. The number of matching lines is automatically logged.

Examples: |
| ${lines} = | Get Lines Containing String | ${result} | An example | | | | ${ret} = | Get Lines Containing String | ${ret} | FAIL | case-insensitive |

See Get Lines Matching Pattern and Get Lines Matching Regexp if you need more complex pattern matching.

get_lines_matching_pattern(string, pattern, case_insensitive=False)
Returns lines of the given string that match the pattern.

The pattern is a _glob pattern_ where: | * | matches everything | | ? | matches any single character | | [chars] | matches any character inside square brackets (e.g. ‘[abc]’ matches either ‘a’, ‘b’ or ‘c’) | | [!chars] | matches any character not inside square brackets | |

A line matches only if it matches the pattern fully. By default the match is case-sensitive, but setting case_insensitive to any value makes it case-insensitive.
Lines are returned as one string concatenated back together with newlines. Possible trailing newline is never returned. The number of matching lines is automatically logged.

Examples: | ${lines} = | Get Lines Matching Pattern | ${result} | Wild???? example | | ${ret} = | Get Lines Matching Pattern | ${ret} | FAIL: * | case-insensitive |

See Get Lines Matching Regexp if you need more complex patterns and Get Lines Containing String if searching literal strings is enough.

`get_lines_matching_regexp(string, pattern)`

Returns lines of the given `string` that match the regexp `pattern`.

See `BuiltIn.Should Match Regexp` for more information about Python regular expression syntax in general and how to use it in Robot Framework test data in particular. A line matches only if it matches the `pattern` fully. Notice that to make the match case-insensitive, you need to embed case-insensitive flag into the `pattern`.

Lines are returned as one string concatenated back together with newlines. Possible trailing newline is never returned. The number of matching lines is automatically logged.

Examples: | ${lines} = | Get Lines Matching Regexp | ${result} | Reg\w{3} example | | ${ret} = | Get Lines Matching Regexp | ${ret} | (?i)FAIL: .* |

See Get Lines Matching Pattern and Get Lines Containing String if you do not need full regular expression powers (and complexity).

`replace_string(string, search_for, replace_with, count=-1)`

Replaces `search_for` in the given `string` with `replace_with`.

`search_for` is used as a literal string. See Replace String Using Regexp if more powerful pattern matching is needed.

If the optional argument `count` is given, only that many occurrences from left are replaced. Negative `count` means that all occurrences are replaced (default behaviour) and zero means that nothing is done.

A modified version of the string is returned and the original string is not altered.

Examples: | ${str} = | Replace String | ${str} | Hello | Hi | | | ${str} = | Replace String | ${str} | world | tellus |

`replace_string_using_regexp(string, pattern, replace_with, count=-1)`

Replaces `pattern` in the given `string` with `replace_with`.

This keyword is otherwise identical to Replace String, but the `pattern` to search for is considered to be a regular expression. See `BuiltIn.Should Match Regexp` for more information about Python regular expression syntax in general and how to use it in Robot Framework test data in particular.

Examples: | ${str} = | Replace String Using Regexp | ${str} | (Hello|Hi) | Hei | | ${str} = | Replace String Using Regexp | ${str} | 20\d\d-\d\d-\d\d | <DATE> | 12 |

`split_string(string, separator=None, max_split=-1)`

Splits the `string` using `separator` as a delimiter string.

If a `separator` is not given, any whitespace string is a separator. In that case also possible consecutive whitespace as well as leading and trailing whitespace is ignored.

Split words are returned as a list. If the optional `max_split` is given, at most `max_split` splits are done, and the returned list will have maximum `max_split + 1` elements.

Examples: | @\{words\} = | Split String | \${string} | | @\{words\} = | Split String | \${string} | $\{SPACE\} | \${pre} | \${post} | | Split String | \${string} | :: | 1 |

See Split String From Right if you want to start splitting from right, and Fetch From Left and Fetch From Right if you only want to get first/last part of the string.
Split the string using separator starting from right.

Same as Split String, but splitting is started from right. This has an effect only when max_split is given.

Examples: | ![split_string_from_right]($\{first\}$) | ![split_string_from_right]($\{others\}$) |
| ![Split String]($\{string\}$) | ![1-1-1] | ![split_string_from_right]($\{others\}$) | ![Split String From Right]($\{string\}$) | ![1-1-1] |

Splits the string to characters.

Example: | ![split_string_to_characters]($\{string\}$) |

Returns contents of the string before the first occurrence of marker.

If the marker is not found, whole string is returned.

See also Fetch From Right, Split String and Split String From Right.

Returns contents of the string after the last occurrence of marker.

If the marker is not found, whole string is returned.

See also Fetch From Left, Split String and Split String From Right.

Generates a string with a desired length from the given chars.

The population sequence chars contains the characters to use when generating the random string. It can contain any characters, and it is possible to use special markers explained in the table below:

| ![LOWER] | Lowercase ASCII characters from 'a' to 'z'. |
| ![UPPER] | Uppercase ASCII characters from 'A' to 'Z'. |
| ![LETTERS] | Lowercase and uppercase ASCII characters. |
| ![NUMBERS] | Numbers from 0 to 9. |

Examples: | ![generate_random_string]($\{ret\}$) | ![generate_random_string]($\{low\}$) | ![generate_random_string]($\{bin\}$) | ![generate_random_string]($\{hex\}$) |

Returns a substring from start index to end index.

The start index is inclusive and end is exclusive. Indexing starts from 0, and it is possible to use negative indices to refer to characters from the end.

Examples: | ![get_substring]($\{ignore first\}$) | ![get_substring]($\{string\}$) | ![1-1-1] | ![get_substring]($\{ignore last\}$) | ![get_substring]($\{string\}$) | ![1-1-1] | ![get_substring]($\{5th to 10th\}$) | ![get_substring]($\{string\}$) | ![1-1-1] | ![get_substring]($\{first two\}$) | ![get_substring]($\{string\}$) | ![1-1-1] | ![get_substring]($\{last two\}$) | ![get_substring]($\{string\}$) | ![1-2-1] |

Fails if the given item is not a string.

This keyword passes regardless is the item is a Unicode string or a byte string. Use Should Be Unicode String or Should Be Byte String if you want to restrict the string type.

The default error message can be overridden with the optional msg argument.
should_not_be_string(item, msg=None)
Fails if the given item is a string.
The default error message can be overridden with the optional msg argument.

should_be_unicode_string(item, msg=None)
Fails if the given item is not a Unicode string.
Use Should Be Byte String if you want to verify the item is a byte string, or Should Be String if both Unicode and byte strings are fine.
The default error message can be overridden with the optional msg argument.
New in Robot Framework 2.7.7.

should_be_byte_string(item, msg=None)
Fails if the given item is not a byte string.
Use Should Be Unicode String if you want to verify the item is a Unicode string, or Should Be String if both Unicode and byte strings are fine.
The default error message can be overridden with the optional msg argument.
New in Robot Framework 2.7.7.

should_be_lowercase(string, msg=None)
Fails if the given string is not in lowercase.
For example ‘string’ and ‘with specials!’ would pass, and ‘String’, ‘ ‘ and ‘ ’ would fail.
The default error message can be overridden with the optional msg argument.
See also Should Be Uppercase and Should Be Titlecase. All these keywords were added in Robot Framework 2.1.2.

should_be_uppercase(string, msg=None)
Fails if the given string is not in uppercase.
For example ‘STRING’ and ‘WITH SPECIALS!’ would pass, and ‘String’, ‘ ‘ and ‘ ‘ would fail.
The default error message can be overridden with the optional msg argument.
See also Should Be Titlecase and Should Be Lowercase. All these keywords were added in Robot Framework 2.1.2.

should_be_titlecase(string, msg=None)
Fails if given string is not title.
string is a titlecased string if there is at least one character in it, uppercase characters only follow uncased characters and lowercase characters only cased ones.
For example ‘This Is Title’ would pass, and ‘Word In UPPER’, ‘Word In lower’, ‘ ‘ and ‘ ‘ would fail.
The default error message can be overridden with the optional msg argument.
See also Should Be Uppercase and Should Be Lowercase. All theses keyword were added in Robot Framework 2.1.2.

4.7.13 Telnet Module

class robot.libraries.Telnet.Telnet (timeout='3 seconds', newline='CRLF', prompt=None, prompt_is_regexp=False, encoding='UTF-8', encoding_errors='ignore', default_log_level='INFO')
A test library providing communication over Telnet connections.
Telnet is Robot Framework’s standard library that makes it possible to connect to Telnet servers and execute commands on the opened connections.

== Table of contents ==
- Connections
- Writing and reading
- Configuration
- Importing
- Logging
- Time string format
- Shortcuts
- Keywords

= Connections =
The first step of using Telnet is opening a connection with Open Connection keyword. Typically the next step is logging in with Login keyword, and in the end the opened connection can be closed with Close Connection.

It is possible to open multiple connections and switch the active one using Switch Connection. Close All Connections can be used to close all the connections, which is especially useful in suite teardowns to guarantee that all connections are always closed.

= Writing and reading =
After opening a connection and possibly logging in, commands can be executed or text written to the connection for other reasons using Write and Write Bare keywords. The main difference between these two is that the former adds a [#Configuration|configurable newline] after the text automatically.

After writing something to the connection, the resulting output can be read using Read, Read Until, Read Until Regexp, and Read Until Prompt keywords. Which one to use depends on the context, but the latest one is often the most convenient.

As a convenience when running a command, it is possible to use Execute Command that simply uses Write and Read Until Prompt internally. Write Until Expected Output is useful if you need to wait until writing something produces a desired output.

Written and read text is automatically encoded/decoded using a [#Configuration|configured encoding].

= Configuration =
Many aspects related the connections can be easily configured either globally or per connection basis. Global configuration is done when [#Importing|library is imported], and these values can be overridden per connection by Open Connection or with setting specific keywords Set Timeout, Set Newline, Set Prompt, Set Encoding, and Set Default Log Level

== Timeout ==
Timeout defines how long is the maximum time to wait when reading output. It is used internally by Read Until, Read Until Regexp, Read Until Prompt, and Login keywords. The default value is 3 seconds.

== Newline ==
Newline defines which line separator Write keyword should use. The default value is CRLF that is typically used by Telnet connections.

Newline can be given either in escaped format using ‘n’ and ‘r’ or with special ‘LF’ and ‘CR’ syntax.

Examples: | Set Newline | n || Set Newline | CRLF |
== Prompt ==

Often the easiest way to read the output of a command is reading all the output until the next prompt with Read Until Prompt. It also makes it easier, and faster, to verify did Login succeed.

Prompt can be specified either as a normal string or a regular expression. The latter is especially useful if the prompt changes as a result of the executed commands.

== Encoding ==

To ease handling text containing non-ASCII characters, all written text is encoded and read text decoded by default. The default encoding is UTF-8 that works also with ASCII. Encoding can be disabled by using a special encoding value NONE. This is mainly useful if you need to get the bytes received from the connection as-is.

Notice that when writing to the connection, only Unicode strings are encoded using the defined encoding. Byte strings are expected to be already encoded correctly. Notice also that normal text in test data is passed to the library as Unicode and you need to use variables to use bytes.

It is also possible to configure the error handler to use if encoding or decoding characters fails. Accepted values are the same that encode/decode functions in Python strings accept. In practice the following values are the most useful:

- ignore: ignore characters that cannot be encoded (default)
- strict: fail if characters cannot be encoded
- replace: replace characters that cannot be encoded with a replacement character

Examples: | Open Connection | localhost | encoding=Latin1 | encoding_errors=strict | | Set Encoding | ISO-8859-15 | Set Encoding | errors=ignore |

Using UTF-8 encoding by default and being able to configure the encoding are new features in Robot Framework 2.7.6. In earlier versions only ASCII was supported and encoding errors were silently ignored. Robot Framework 2.7.7 added a possibility to specify the error handler, changed the default behavior back to ignoring encoding errors, and added the possibility to disable encoding.

== Default log level ==

Default log level specifies the log level keywords use for logging unless they are given an explicit log level. The default value is INFO, and changing it, for example, to DEBUG can be a good idea if there is lot of unnecessary output that makes log files big.

Configuring default log level in importing and with Open Connection are new features in Robot Framework 2.7.6. In earlier versions only Set Default Log Level could be used.

= Logging =

All keywords that read something log the output. These keywords take the log level to use as an optional argument, and if no log level is specified they use the [Configuration]configured default value.

The valid log levels to use are TRACE, DEBUG, INFO (default), and WARN. Levels below INFO are not shown in log files by default whereas warnings are shown more prominently.

The [http://docs.python.org/2/library/telnetlib.html#telnetlib module] used by this library has a custom logging system for logging content it sends and receives. Starting from Robot Framework 2.7.7, these low level log messages are forwarded to Robot's log file using TRACE level.

= Time string format =

Timeouts and other times used must be given as a time string using format like ‘15 seconds’ or ‘1min 10s’. If the timeout is given as just a number, for example, ‘10’ or ‘1.5’, it is considered to be seconds. The time string format is described in more detail in an appendix of [http://code.google.com/p/robotframework/wiki/UserGuideRobot Framework User Guide].
Telnet library can be imported with optional configuration parameters.

Configuration parameters are used as default values when new connections are opened with Open Connection keyword. They can also be overridden after opening the connection using the Set Timeout, Set Newline, Set Prompt, Set Encoding, and Set Default Log Level keywords. See these keywords and Configuration section above for more information about these parameters and their possible values.

Examples (use only one of these):

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
<th>Value</th>
<th>Value</th>
<th>Value</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library</td>
<td>Telnet</td>
<td></td>
<td></td>
<td></td>
<td># default values</td>
</tr>
<tr>
<td>Library</td>
<td>Telnet</td>
<td>0.5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Library</td>
<td>Telnet</td>
<td>newline=LF</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Library</td>
<td>Telnet</td>
<td>newline=LF</td>
<td>encoding=ISO-8859-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Library</td>
<td>Telnet</td>
<td>2.0</td>
<td>LF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Library</td>
<td>Telnet</td>
<td>2.0</td>
<td>CRLF</td>
<td>$</td>
<td></td>
</tr>
<tr>
<td>Library</td>
<td>Telnet</td>
<td>2.0</td>
<td>LF</td>
<td>(&gt;</td>
<td># )</td>
</tr>
</tbody>
</table>

ROBOT_LIBRARY_SCOPE = 'TEST_SUITE'

ROBOT_LIBRARY_VERSION = '2.7.7'

get_keyword_names()

open_connection (host, alias=None, port=23, timeout=None, newline=None, prompt=None, prompt_is_regexp=False, encoding=None, encoding_errors=None, default_log_level=None)

Opens a new Telnet connection to the given host and port.

The timeout, newline, prompt, prompt_is_regexp, encoding, and default_log_level arguments get default values when the library is imported. Setting them here overrides those values for the opened connection. See Configuration section for more information.

Possible already opened connections are cached and it is possible to switch back to them using Switch Connection keyword. It is possible to switch either using explicitly given alias or using index returned by this keyword. Indexing starts from 1 and is reset back to it by Close All Connections keyword.

switch_connection (index_or_alias)

Switches between active connections using an index or an alias.

Aliases can be given to Open Connection keyword which also always returns the connection index.

This keyword returns the index of previous active connection.

Example: | Open Connection | myhost.net | | | | Login | john | secret | Write | some command | | Open Connection | yourhost.com | 2nd conn | | Login | root | password | Write | another cmd | | index | Write | something | Switch Connection | 2nd conn | | alias | Write | whatever | Switch Connection | 2 | back to original | | [Teardown] | Close All Connections |

The example above expects that there were no other open connections when opening the first one, because it used index ‘1’ when switching to the connection later. If you are not sure about that, you can store the index into a variable as shown below.

${index} = | Open Connection | myhost.net | | | | Do Something |
Switch Connection | ${index} |

close_all_connections()
Closes all open connections and empties the connection cache.

If multiple connections are opened, this keyword should be used in a test or suite teardown to make sure
that all connections are closed. It is not an error if some of the connections have already been closed by
Close Connection.

After this keyword, new indexes returned by Open Connection keyword are reset to 1.

class robot.libraries.Telnet.TelnetConnection (host=None, port=23, timeout=3.0,
newline='CRLF', prompt=None,
prompt_is_regexp=False, encoding='UTF-8', encoding_errors='ignore',
default_log_level='INFO')

Bases: telnetlib.Telnet

set_timeout (timeout)
Sets the timeout used for waiting output in the current connection.

Read operations that expect some output to appear (Read Until, Read Until Regexp, Read Until Prompt,
Login) use this timeout and fail if the expected output does not appear before this timeout expires.

The timeout must be given in time string format. The old timeout is returned and can be used to restore the
timeout later.

Example: | ${old} = | Set Timeout | 2 minute 30 seconds | Do Something | Set Timeout | ${old} |

See Configuration section for more information about global and connection specific configuration.

set_newline (newline)
Sets the newline used by Write keyword in the current connection.

The old newline is returned and can be used to restore the newline later. See Set Timeout for a similar
eexample.

See Configuration section for more information about global and connection specific configuration.

set_prompt (prompt, prompt_is_regexp=False)
Sets the prompt used by Read Until Prompt and Login in the current connection.

If prompt_is_regexp is given any true value, including any non-empty string, the given prompt is considered
to be a regular expression.

The old prompt is returned and can be used to restore the prompt later.

Example: | ${prompt} | ${regexp} = | Set Prompt | Do Something | Set Prompt | ${prompt} | ${regexp} |

See the documentation of [http://docs.python.org/2/library/re.htmlPython re module] for more information
about the supported regular expression syntax. Notice that possible backslashes need to be escaped in
Robot Framework test data.

See Configuration section for more information about global and connection specific configuration.

set_encoding (encoding=None, errors=None)
Sets the encoding to use for writing and reading in the current connection.

The given encoding specifies the encoding to use when written/read text is encoded/decoded, and errors
specifies the error handler to use if encoding/decoding fails. Either of these can be omitted and in that case
the old value is not affected. Use string NONE to disable encoding altogether.
See Configuration section for more information about encoding and error handlers, as well as global and connection specific configuration in general.

The old values are returned and can be used to restore the encoding and the error handler later. See Set Prompt for a similar example.

Setting encoding in general is a new feature in Robot Framework 2.7.6. Specifying the error handler and disabling encoding were added in 2.7.7.

set_default_log_level(level)
Sets the default log level used for logging in the current connection.

The old default log level is returned and can be used to restore the log level later.

See Configuration section for more information about global and connection specific configuration.

close_connection(loglevel=None)
Closes the current Telnet connection.

Remaining output in the connection is read, logged, and returned. It is not an error to close an already closed connection.

Use Close All Connections if you want to make sure all opened connections are closed.

See Logging section for more information about log levels.

login(username, password, login_prompt='login: ', password_prompt='Password: ', login_timeout='1 second', login_incorrect='Login incorrect')
Logs in to the Telnet server with the given user information.

This keyword reads from the connection until the login_prompt is encountered and then types the given username. Then it reads until the password_prompt and types the given password. In both cases a newline is appended automatically and the connection specific timeout used when waiting for outputs.

How logging status is verified depends on whether a prompt is set for this connection or not:

1) If the prompt is set, this keyword reads the output until the prompt is found using the normal timeout. If no prompt is found, login is considered failed and also this keyword fails. Note that in this case both login_timeout and login_incorrect arguments are ignored.

2) If the prompt is not set, this keywords sleeps until login_timeout and then reads all the output available on the connection. If the output contains login_incorrect text, login is considered failed and also this keyword fails. Both of these configuration parameters were added in Robot Framework 2.7.6. In earlier versions they were hard coded.

See Configuration section for more information about setting newline, timeout, and prompt.

write(text, loglevel=None)
Writes the given text plus a newline into the connection.

The newline character sequence to use can be [#Configuration|configured] both globally and per connection basis. The default value is CRLF.

This keyword consumes the written text, until the added newline, from the output and logs and returns it. The given text itself must not contain newlines. Use Write Bare instead if either of these features causes a problem.

Note: This keyword does not return the possible output of the executed command. To get the output, one of the Read ... keywords must be used. See Writing and reading section for more details.

See Logging section for more information about log levels.

write_bare(text)
Writes the given text, and nothing else, into the connection.
This keyword does not append a newline nor consume the written text. Use Write if these features are needed.

**write_until_expected_output** *(text, expected, timeout, retry_interval, loglevel=None)*

Writes the given *text* repeatedly, until *expected* appears in the output.

*text* is written without appending a newline and it is consumed from the output before trying to find *expected*. If *expected* does not appear in the output within *timeout*, this keyword fails.

*retry_interval* defines the time to wait *expected* to appear before writing the *text* again. Consuming the written *text* is subject to the normal [#Configuration|configured timeout].

Both *timeout* and *retry_interval* must be given in *time string format*. See Logging section for more information about log levels.

Example: | Write Until Expected Output | ps -ef | grep myprocessrn | myprocess | | ... | 5 s | 0.5 s |

The above example writes command *ps -ef | grep myprocessrn* until *myprocess* appears in the output. The command is written every 0.5 seconds and the keyword fails if *myprocess* does not appear in the output in 5 seconds.

**read** *(loglevel=None)*

Reads everything that is currently available in the output.

Read output is both returned and logged. See Logging section for more information about log levels.

**read_until** *(expected, loglevel=None)*

Reads output until *expected* text is encountered.

Text up to and including the match is returned and logged. If no match is found, this keyword fails. How much to wait for the output depends on the [#Configuration|configured timeout].

See Logging section for more information about log levels. Use Read Until Regexp if more complex matching is needed.

**read_until_regexp** *(expected)*

Reads output until any of the *expected* regular expressions match.

This keyword accepts any number of regular expressions patterns or compiled Python regular expression objects as arguments. Text up to and including the first match to any of the regular expressions is returned and logged. If no match is found, this keyword fails. How much to wait for the output depends on the [#Configuration|configured timeout].

If the last given argument is a [#Logging|valid log level], it is used as *loglevel* similarly as with Read Until keyword.

See the documentation of [http://docs.python.org/2/library/re.html|Python re module] for more information about the supported regular expression syntax. Notice that possible backslashes need to be escaped in Robot Framework test data.

Examples: | Read Until Regexp | (#$) | Read Until Regexp | first_regexp | second_regexp | Read Until Regexp | \d{4}-\d{2}-\d{2} | DEBUG |

**read_until_prompt** *(loglevel=None)*

Reads output until the prompt is encountered.

This keyword requires the prompt to be [#Configuration|configured] either in importing or with Open Connection or Set Prompt keyword.

Text up to and including the prompt is returned and logged. If no prompt is found, this keyword fails. How much to wait for the output depends on the [#Configuration|configured timeout].

See Logging section for more information about log levels.
execute_command(command, loglevel=None)

Executes the given command and reads, logs, and returns everything until the prompt.

This keyword requires the prompt to be [#Configuration|configured] either in importing or with Open Connection or Set Prompt keyword.

This is a convenience keyword that uses Write and Read Until Prompt internally Following two examples are thus functionally identical:

$$\text{\$\{out\} = |Execute Command| pwd |}$$

Write | pwd |
$\text{\$\{out\} = |Read Until Prompt|}$$

See Logging section for more information about log levels.

msg(msg, *args)

close()

Close the connection.

expect(list, timeout=None)

Read until one from a list of regular expressions matches.

The first argument is a list of regular expressions, either compiled (re.RegexObject instances) or uncompiled (strings). The optional second argument is a timeout, in seconds; default is no timeout.

Return a tuple of three items: the index in the list of the first regular expression that matches; the match object returned; and the text read up till and including the match.

If EOF is read and no text was read, raise EOFError. Otherwise, when nothing matches, return (-1, None, text) where text is the text received so far (may be the empty string if a timeout happened).

If a regular expression ends with a greedy match (e.g. '.*') or if more than one expression can match the same input, the results are undeterministic, and may depend on the I/O timing.

fileno()

Return the fileno() of the socket object used internally.

fill_rawq()

Fill raw queue from exactly one recv() system call.

Block if no data is immediately available. Set self.eof when connection is closed.

get_socket()

Return the socket object used internally.

interact()

Interaction function, emulates a very dumb telnet client.

listener()

Helper for mt_interact() – this executes in the other thread.

mt_interact()

Multithreaded version of interact().

open(host, port=0, timeout=<object object at 0x7f0990b40240>)

Connect to a host.

The optional second argument is the port number, which defaults to the standard telnet port (23).
Don’t try to reopen an already connected instance.

`process_rawq()`  
Transfer from raw queue to cooked queue.
Set self.eof when connection is closed. Don’t block unless in the midst of an IAC sequence.

`rawq_getchar()`  
Get next char from raw queue.
Block if no data is immediately available. Raise EOFError when connection is closed.

`read_all()`  
Read all data until EOF; block until connection closed.

`read_eager()`  
Read readily available data.
Raise EOFError if connection closed and no cooked data available. Return ‘’ if no cooked data available otherwise. Don’t block unless in the midst of an IAC sequence.

`read_lazy()`  
Process and return data that’s already in the queues (lazy).
Raise EOFError if connection closed and no data available. Return ‘’ if no cooked data available otherwise. Don’t block unless in the midst of an IAC sequence.

`read_sb_data()`  
Return any data available in the SB ... SE queue.
Return ‘’ if no SB ... SE available. Should only be called after seeing a SB or SE command. When a new SB command is found, old unread SB data will be discarded. Don’t block.

`read_some()`  
Read at least one byte of cooked data unless EOF is hit.
Return ‘’ if EOF is hit. Block if no data is immediately available.

`read_very_eager()`  
Read everything that’s possible without blocking in I/O (eager).
Raise EOFError if connection closed and no cooked data available. Return ‘’ if no cooked data available otherwise. Don’t block unless in the midst of an IAC sequence.

`read_very_lazy()`  
Return any data available in the cooked queue (very lazy).
Raise EOFError if connection closed and no data available. Return ‘’ if no cooked data available otherwise. Don’t block.

`set_debuglevel(debuglevel)`  
Set the debug level.
The higher it is, the more debug output you get (on sys.stdout).

`set_option_negotiation_callback(callback)`  
Provide a callback function called after each receipt of a telnet option.

`sock_avail()`  
Test whether data is available on the socket.

4.7. libraries Package
4.7.14 XML Module

class robot.libraries.XML.XML
Bases: object

Robot Framework test library for verifying and modifying XML documents.

As the name implies, XML is a test library for verifying contents of XML files. In practice it is a pretty thin wrapper on top of Python’s [http://docs.python.org/library/xml.etree.elementtree.html|ElementTree XML API].

The library has the following main usages:

• Parsing an XML file, or a string containing XML, into an XML element structure and finding certain elements from it for further analysis (e.g. Parse XML and Get Element keywords).
• Getting text or attributes of elements (e.g. Get Element Text and Get Element Attribute).
• Directly verifying text, attributes, or whole elements (e.g Element Text Should Be and Elements Should Be Equal).
• Modifying XML and saving it (e.g. Set Element Text, Add Element and Save XML).

== Table of contents ==
• Parsing XML
• Example
• Finding elements with xpath
• Element attributes
• Handling XML namespaces
• Shortcuts
• Keywords

= Parsing XML =

XML can be parsed into an element structure using Parse XML keyword. It accepts both paths to XML files and strings that contain XML. The keyword returns the root element of the structure, which then contains other elements as its children and their children.

The element structure returned by Parse XML, as well as elements returned by keywords such as Get Element, can be used as the source argument with other keywords. In addition to an already parsed XML structure, other keywords also accept paths to XML files and strings containing XML similarly as Parse XML.

When the source is given as a path to a file, the forward slash character (/) can be used as the path separator regardless the operating system. On Windows also the backslash works, but it the test data it needs to be escaped by doubling it (\). Using the built-in variable ${/} naturally works too.

= Example =

The following simple example demonstrates parsing XML and verifying its contents both using keywords in this library and in Builtin and Collections libraries. How to use xpath expressions to find elements and what attributes the returned elements contain are discussed, with more examples, in Finding elements with xpath and Element attributes sections.

In this example, as well as in many other examples in this documentation, ${XML} refers to the following example XML document. In practice ${XML} could either be a path to an XML file or it could contain the XML itself.

<example>
Notice that in the example three last lines are equivalent. Which one to use in practice depends on which other elements you need to get or verify. If you only need to do one verification, using the last line alone would suffice. If more verifications are needed, parsing the XML with `Parse XML` only once would be more efficient.

= Finding elements with xpath =

ElementTree, and thus also this library, supports finding elements using xpath expressions. ElementTree does not, however, support the full xpath syntax, and what is supported depends on its version. ElementTree 1.3 that is distributed with Python and Jython 2.7 supports richer syntax than versions distributed with earlier Python interpreters.

The supported xpath syntax is explained below and [http://effbot.org/zone/element-xpath.htm|ElementTree documentation] provides more details. In the examples `$XML` refers to the same XML structure as in the earlier example.

== Tag names ==

When just a single tag name is used, xpath matches all direct child elements that have that tag name.

```robot
$root = Parse XML | $XML | Should Be Equal | $root.tag | example |
$first = Get Element | $root | first | Should Be Equal | $first.text | Text |
Dictionary Should Contain Key | $first.attrib | id | Element Text Should Be | $first | text |
Element Attribute Should Be | $first | id | Element Attribute Should Be | $root | id | xpath=first |
Element Attribute Should Be | $XML | id | xpath=first |
```

```robot
$elem = Get Element | $XML | third |
Should Be Equal | $elem.tag | third |
@children = Get Elements | $elem | child |
Length Should Be | $children | 2 |
```
== Paths ==

Paths are created by combining tag names with a forward slash (/). For example, parent/child matches all child elements under parent element. Notice that if there are multiple parent elements that all have child elements, parent/child xpath will match all these child elements.

```robotframework
${elem} = Get Element ${XML} second/child
Should Be Equal ${elem.tag} child
${elem} = Get Element ${XML} third/child/grandchild
Should Be Equal ${elem.tag} grandchild
```

== Wildcards ==

An asterisk (*) can be used in paths instead of a tag name to denote any element.

```robotframework
@{children} = Get Elements ${XML} */child
Length Should Be ${children} 3
```

== Current element ==

The current element is denoted with a dot (.). Normally the current element is implicit and does not need to be included in the xpath.

== Parent element ==

The parent element of another element is denoted with two dots (..). Notice that it is not possible to refer to the parent of the current element. This syntax is supported only in ElementTree 1.3 (i.e. Python/Jython 2.7 and newer).

```robotframework
${elem} = Get Element ${XML} */second/..  
Should Be Equal ${elem.tag} third
```

== Search all sub elements ==

Two forward slashes (//=) mean that all sub elements, not only the direct children, are searched. If the search is started from the current element, an explicit dot is required.

```robotframework
@{elements} = Get Elements ${XML} //second
Length Should Be ${elements} 2
${b} = Get Element ${XML} html//b
Should Be Equal ${b.text} bold
```

== Predicates ==

Predicates allow selecting elements using also other criteria than tag names, for example, attributes or position. They are specified after the normal tag name or path using syntax path[predicate]. The path can have wildcards and other special syntax explained above.
What predicates ElementTree supports is explained in the table below. Notice that predicates in general are supported only in ElementTree 1.3 (i.e. Python/Jython 2.7 and newer).

<table>
<thead>
<tr>
<th><em>Predicate</em></th>
<th><em>Matches</em></th>
<th><em>Example</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>@attrib</td>
<td>Elements with attribute <code>attrib</code>.</td>
<td><code>second[@id]</code></td>
</tr>
<tr>
<td>@attrib=&quot;value&quot;</td>
<td>Elements with attribute <code>attrib</code> having value <code>value</code>.</td>
<td><code>[@id=&quot;2&quot;]</code></td>
</tr>
<tr>
<td>position</td>
<td>Elements at the specified position. Position can be an integer (starting from 1), expression <code>last()</code>, or relative expression like <code>last() - 1</code>.</td>
<td><code>third/child[1]</code></td>
</tr>
<tr>
<td>tag</td>
<td>Elements with a child element named <code>tag</code>.</td>
<td><code>third/child[grandchild]</code></td>
</tr>
</tbody>
</table>

Predicates can also be stacked like `path[predicate1][predicate2]`. A limitation is that possible position predicate must always be first.

= Element attributes =

All keywords returning elements, such as `Parse XML`, and `Get Element`, return ElementTree’s [http://docs.python.org/library/xml.etree.elementtree.html#xml.etree.ElementTree.ElementTree.ElementTree.Element objects]. These elements can be used as inputs for other keywords, but they also contain several useful attributes that can be accessed directly using the extended variable syntax.

The attributes that are both useful and convenient to use in the test data are explained below. Also other attributes, including methods, can be accessed, but that is typically better to do in custom libraries than directly in the test data.

The examples use the same `$/XML/` structure as the earlier examples.

== tag ==
The tag of the element.

```
${root} = | Parse XML | ${XML} |
Should Be Equal | ${root.tag} | example |
```

== text ==
The text that the element contains or Python `None` if the element has no text. Notice that the text _does not_ contain texts of possible child elements nor text after or between children. Notice also that in XML whitespace is significant, so the text contains also possible indentation and newlines. To get also text of the possible children, optionally whitespace normalized, use `Get Element Text` keyword.

```
${1st} = | Get Element | ${XML} |
Should Be Equal | ${1st.text} | text |

${2nd} = | Get Element | ${XML} |
Should Be Equal | ${2nd.text} | ${NONE} |

${p} = | Get Element | ${XML} |
Should Be Equal | ${p.text} | n${SPACE*6}Text with${SPACE} |
```

== tail ==
The text after the element before the next opening or closing tag. Python None if the element has no tail. Similarly as with text, also tail contains possible indentation and newlines.

\$\{b\} = \|Get Element\|  \$\{XML\}\| html/p/b\|
\textit{Should Be Equal} \|  \$\{b.tail\}\|  \$\{SPACE\}and\$\{SPACE\}\|

== attrib ==

A Python dictionary containing attributes of the element.

\$\{2nd\} = \|Get Element\|  \$\{XML\}\| second\|
\textit{Should Be Equal} \|  \$\{2nd.attrib[\textquoteleft id\textquoteright]\}\|  2\|
\$\{3rd\} = \|Get Element\|  \$\{XML\}\| third\|
\textit{Should Be Empty} \|  \$\{3rd.attrib\}\|  1\|

= Handling XML namespaces =

ElementTree handles possible namespaces in XML documents by adding the namespace URI to tag names in so-called Clark Notation. That is inconvenient especially with xpaths, and by default this library strips those namespaces away and moves them to xmlns attribute instead. That can be avoided by passing keep_clark_notation argument to \textit{Parse XML} keyword. The pros and cons of both approaches are discussed in more detail below.

== How ElementTree handles namespaces ==

If an XML document has namespaces, ElementTree adds namespace information to tag names in [http://www.jclark.com/xml/xmlns.html|Clark Notation] (e.g. \{http://ns.uri\}tag and removes original xmlns attributes. This is done both with default namespaces and with namespaces with a prefix. How it works in practice is illustrated by the following example, where \$\{NS\} variable contains this XML document:

\<xsl:stylesheet xmlns:xsl=\"http://www.w3.org/1999/XSL/Transform\"
        xmlns=\"http://www.w3.org/1999/xhtml\">
  \<xsl:template match=\"/\">
    \<html>\</html>
  \</xsl:template>
\</xsl:stylesheet>

\$\{root\} = \|Parse XML\|  \$\{NS\}  keep_clark_notation=yes\|
\textit{Should Be Equal} \|  \$\{root.tag\}\|  \{http://www.w3.org/1999/XSL/Transform\}stylesheet\|
\textit{Element Should Exist} \|  \$\{root\}\|
\{http://www.w3.org/1999/XSL/Transform\}template/\{http://www.w3.org/1999/xhtml\}html\|
\textit{Should Be Empty} \|  \$\{root.attrib\}\|

As you can see, including the namespace URI in tag names makes xpaths really long and complex.

If you save the XML, ElementTree moves namespace information back to xmlns attributes. Unfortunately it does not restore the original prefixes:
The resulting output is semantically same as the original, but mangling prefixes like this may still not be desirable. Notice also that the actual output depends slightly on ElementTree version.

== Default namespace handling ==

Because the way ElementTree handles namespaces makes xpaths so complicated, this library, by default, strips namespaces from tag names and moves that information back to xmlns attributes. How this works in practice is shown by the example below, where ${NS}$ variable contains the same XML document as in the previous example.

```
${root} = | Parse XML | ${NS} | Should Be Equal | ${root.tag} | stylesheet |
Element Should Exist | ${root} | template/html |
Element Attribute Should Be | ${root} | xmlns | http://www.w3.org/1999/XSL/Transform |
Element Attribute Should Be | ${root} | xmlns | http://www.w3.org/1999/xhtml | xpath=template/html |
```

Now that tags do not contain namespace information, xpaths are simple again.

A minor limitation of this approach is that namespace prefixes are lost. As a result the saved output is not exactly same as the original one in this case either:

```
<stylesheet xmlns="http://www.w3.org/1999/XSL/Transform">
    <template match="/>
        <html xmlns="http://www.w3.org/1999/xhtml"></html>
    </template>
</stylesheet>
```

Also this output is semantically same as the original. If the original XML had only default namespaces, the output would also looks identical.

== Attribute namespaces ==

Attributes in XML documents are, by default, in the same namespaces as the element they belong to. It is possible to use different namespaces by using prefixes, but this is pretty rare.

If an attribute has a namespace prefix, ElementTree will replace it with Clark Notation the same way it handles elements. Because stripping namespaces from attributes could cause attribute conflicts, this library does not handle attribute namespaces at all. Thus the following example works the same way regardless how namespaces are handled.

```
${root} = | Parse XML | <root id="1" ns:id="2" xmlns:ns="http://my.ns"/> | Element Attribute Should Be | ${root} | id | 1 |
Element Attribute Should Be | ${root} | {http://my.ns}id | 2 |
```
parse_xml(source, keep_clark_notation=False)

Parses the given XML file or string into an element structure.

The `source` can either be a path to an XML file or a string containing XML. In both cases the XML is parsed into ElementTree [http://docs.python.org/library/xml.etree.elementtree.html#xml.etree.ElementTree.ElementTree](http://docs.python.org/library/xml.etree.elementtree.html#xml.etree.ElementTree.ElementTree) structure and the root element is returned.

As discussed in the Handling XML namespaces section, this keyword, by default, strips possible namespaces added by ElementTree into tag names. This typically eases handling XML documents with namespaces considerably. If you do not want that to happen, or want to avoid the small overhead of going through the element structure when your XML does not have namespaces, you can disable this feature by giving `keep_clark_notation` argument a true value (e.g. any non-empty string).

Examples:

```robot
$root = Parse XML <root><child/></root>
```

```robot
$xml = Parse XML ${CUR-DIR}/test.xml no namespace cleanup
```

Use `Get Element` keyword if you want to get a certain element and not the whole structure. See Parsing XML section for more details and examples.

The keyword fails if more, or less, than one element matches the `xpath`. Use `Get Elements` if you want all matching elements to be returned.

Examples using ${XML} structure from the introduction:

```robot
${element} = Get Element ${XML} second
${child} = Get Element ${element} child
```

`Parse XML` is recommended for parsing XML when the whole structure is needed. It must be used if there is a need to configure how XML namespaces are handled.

get_element(source, xpath='.')

Returns an element in the `source` matching the `xpath`.

The `source` can be a path to an XML file, a string containing XML, or an already parsed XML element. The `xpath` specifies which element to find. See the introduction for more details about both the possible sources and the supported `xpath` syntax.

The keyword fails if more, or less, than one element matches the `xpath`. Use `Get Elements` if you want all matching elements to be returned.

Examples using ${XML} structure from the introduction:

```robot
${element} = Get Element ${XML} element 1
${child} = Get Element ${element} child
```

Use `Get Elements` if you want all matching elements to be returned.

The keyword fails if more, or less, than one element matches the `xpath`. Use `Get Elements` if you want all matching elements to be returned.

Examples using ${XML} structure from the introduction:

```robot
${children} = Get Elements ${XML} third/child |
Length Should Be | ${children} | 2 |
${children} = Get Elements ${XML} first/child |
Should Be Empty | ${children} |
```

get_child_elements(source, xpath='.')

Returns the child elements of the specified element as a list.

The element whose children to return is specified using `source` and `xpath`. They have exactly the same semantics as with `Get Element` keyword.

All the direct child elements of the specified element are returned. If the element has no children, an empty list is returned.
Examples using ${XML} structure from the introduction:

```robot
| {children} = Get Child Elements | {XML} |
| Length Should Be | {children} | 4 |
| Should Be Empty | {children} |
```

**get_element_count** ([source], xpath='.')

Returns and logs how many elements the given xpath matches.

Arguments `source` and `xpath` have exactly the same semantics as with `Get Elements` keyword that this keyword uses internally.

See also `Element Should Exist` and `Element Should Not Exist`.

New in Robot Framework 2.7.5.

**element_should_exist** ([source], xpath='.', message=None)

Verifies that one or more element match the given xpath.

Arguments `source` and `xpath` have exactly the same semantics as with `Get Elements` keyword. Keyword passes if the `xpath` matches one or more elements in the `source`. The default error message can be overridden with the `message` argument.

See also `Element Should Not Exist` as well as `Get Element Count` that this keyword uses internally.

New in Robot Framework 2.7.5.

**element_should_not_exist** ([source], xpath='.', message=None)

Verifies that no element match the given xpath.

Arguments `source` and `xpath` have exactly the same semantics as with `Get Elements` keyword. Keyword fails if the `xpath` matches any element in the `source`. The default error message can be overridden with the `message` argument.

See also `Element Should Exist` as well as `Get Element Count` that this keyword uses internally.

New in Robot Framework 2.7.5.

**get_element_text** ([source], xpath='.', normalize_whitespace=False)

Returns all text of the element, possibly whitespace normalized.

The element whose text to return is specified using `source` and `xpath`. They have exactly the same semantics as with `Get Element` keyword.

This keyword returns all the text of the specified element, including all the text its children and grandchildren contains. If the element has no text, an empty string is returned. As discussed in the introduction, the returned text is thus not always the same as the text attribute of the element.

Be default all whitespace, including newlines and indentation, inside the element is returned as-is. If `normalize_whitespace` is given any true value (e.g. any non-empty string), then leading and trailing whitespace is stripped, newlines and tabs converted to spaces, and multiple spaces collapsed into one. This is especially useful when dealing with HTML data.

Examples using ${XML} structure from the introduction:

```robot
| {text} = Get Element Text | {XML} |
| Should Be Equal | {text} |
```

See also `Get Elements Texts`, `Element Text Should Be` and `Element Text Should Match`.

**get_elements_texts** ([source], xpath, normalize_whitespace=False)

Returns text of all elements matching `xpath` as a list.

The elements whose text to return is specified using `source` and `xpath`. They have exactly the same semantics as with `Get Elements` keyword.
The text of the matched elements is returned using the same logic as with \textit{Get Element Text}. This includes optional whitespace normalization using the \texttt{normalize_whitespace} option.

Examples using \texttt{$\{XML\}$ structure from the introduction: \texttt{\{texts\} = \{Get Elements Texts $\{XML\}$ third/child \{Length Should Be \{texts\} 2 \{Should Be Equal \{texts\}[0] more text \{Should Be Equal \{texts\}[1] $\{EMPTY\}$ \}}}

\texttt{element_text_should_be (source, expected, xpath=':', normalize_whitespace=False, message=None)}

Verifies that the text of the specified element is \texttt{expected}.

The element whose text is verified is specified using \texttt{source} and \texttt{xpath}. They have exactly the same semantics as with \textit{Get Element} keyword.

The text to verify is got from the specified element using the same logic as with \textit{Get Element Text}. This includes optional whitespace normalization using the \texttt{normalize_whitespace} option.

The keyword passes if the text of the element is equal to the \texttt{expected} value, and otherwise it fails. The default error message can be overridden with the \texttt{message} argument. Use \texttt{Element Text Should Match} to verify the text against a pattern instead of an exact value.

Examples using \texttt{$\{XML\}$ structure from the introduction: \texttt{\{texts\} = \{Get Elements Texts $\{XML\}$ \texttt{ \{Get Element $\{XML\}$ xpath=first $\{Element Text Should Be $\{XML\} \} \texttt{ \{Get Element $\{XML\}$ xpath=second/child $\{paragraph\} = $\{Get Element $\{XML\}$ xpath=html/p $\{Element Text Should Be $\{paragraph\} $\{Text with bold and italics. \}}} \texttt{\{normalize_whitespace=yes \}} \texttt{\}}}}}

\texttt{element_text_should_match (source, pattern, xpath=':'. normalize_whitespace=False, message=None)}

Verifies that the text of the specified element matches \texttt{expected}.

This keyword works exactly like \textit{Element Text Should Be} except that the expected value can be given as a pattern that the text of the must match.

Pattern matching is similar as matching files in a shell, and it is always case-sensitive. In the pattern, \texttt{*} matches anything and \texttt{?} matches any single character.

Examples using \texttt{$\{XML\}$ structure from the introduction: \texttt{\{texts\} = \{Get Elements Texts $\{XML\}$ \texttt{ $\{paragraph\} = $\{Get Element $\{XML\}$ xpath=html/p $\{Element Text Should Match $\{paragraph\} \} \texttt{Text with * and *. \}}} \texttt{\{normalize_whitespace=yes \}} \texttt{\}}}

\texttt{get_element_attribute (source, name, xpath=':', default=None)}

Returns the named attribute of the specified element.

The element whose attribute to return is specified using \texttt{source} and \texttt{xpath}. They have exactly the same semantics as with \textit{Get Element} keyword.

The value of the attribute \texttt{name} of the specified element is returned. If the element does not have such element, the \texttt{default} value is returned instead.

Examples using \texttt{$\{XML\}$ structure from the introduction: \texttt{\{texts\} = \{Get Element Attribute $\{XML\}$ id \{xpath=first \{Should Be Equal \{attribute\} \{\} \texttt{\}} \texttt{\}} \texttt{\}}}

\texttt{get_element_attributes (source, xpath=':')} Returns all attributes of the specified element.

The element whose attributes to return is specified using \texttt{source} and \texttt{xpath}. They have exactly the same semantics as with \textit{Get Element} keyword.
Attributes are returned as a Python dictionary. It is a copy of the original attributes so modifying it has no effect on the XML structure.

Examples using $\{XML\}$ structure from the introduction:

```
| ${attributes} = | Get Element Attributes |
| $\{XML\}$ | first | | Dictionary Should Contain Key | $\{attributes\}$ | id | | $\{attributes\} = | Get Element Attributes |
| $\{XML\}$ | third | | Should Be Empty | $\{attributes\}$ |
```

Use Get Element Attribute to get the value of a single attribute.

### element_attribute_should_be

Declares that the specified attribute is expected.

```
element_attribute_should_be(source, name, expected, xpath='.', message=None)
```

- **source**: The element whose attribute is verified is specified using `source` and `xpath`. They have exactly the same semantics as with `Get Element` keyword.
- **name**: The keyword passes if the attribute `name` of the element is equal to the `expected` value, and otherwise it fails. The default error message can be overridden with the `message` argument.

To test that the element does not have a certain attribute, Python `None` (i.e. variable `$\{\text{NONE}\}$`) can be used as the `expected` value. A cleaner alternative is using `Element Should Not Have Attribute`.

Examples using $\{XML\}$ structure from the introduction:

```
| Element Attribute Should Be | $\{XML\}$ | id | | Element Attribute Should Be | $\{XML\}$ | id | $\{\text{NONE}\}$ |
```

See also `Element Attribute Should Match` and `Get Element Attribute`.

### element_attribute_should_match

Verifies that the specified attribute matches expected.

```
element_attribute_should_match(source, name, pattern, xpath='.', message=None)
```

- **source**: This keyword works exactly like `Element Attribute Should Be` except that the expected value can be given as a pattern that the attribute of the element must match.

Pattern matching is similar as matching files in a shell, and it is always case-sensitive. In the pattern, `*` matches anything and `?` matches any single character.

Examples using $\{XML\}$ structure from the introduction:

```
| Element Attribute Should Match | $\{XML\}$ | id | | Element Attribute Should Match | $\{XML\}$ | id | $c^d$ | xpath=third/second |
```

### element_should_not_have_attribute

Verifies that the specified element does not have attribute `name`.

```
element_should_not_have_attribute(source, name, xpath='.', message=None)
```

- **source**: The element whose attribute is verified is specified using `source` and `xpath`. They have exactly the same semantics as with `Get Element` keyword.

The keyword fails if the specified element has attribute `name`. The default error message can be overridden with the `message` argument.

Examples using $\{XML\}$ structure from the introduction:

```
| Element Should Not Have Attribute | $\{XML\}$ | id | | Element Should Not Have Attribute | $\{XML\}$ | xxx | xpath=first |
```

See also `Get Element Attribute`, `Get Element Attributes`, `Element Text Should Be` and `Element Text Should Match`.

New in Robot Framework 2.7.5.

### elements_should_be_equal

Verifies that the given `source` element is equal to `expected`.

```
elementsshould_be_equal(source, expected, exclude_children=False, normalize_whitespace=False)
```

- **source** and **expected** can be given as a path to an XML file, as a string containing XML, or as an already parsed XML element structure. See `introduction` for more information about parsing XML in general.
The keyword passes if the source element and expected element are equal. This includes testing the tag names, texts, and attributes of the elements. By default also child elements are verified the same way, but this can be disabled by setting exclude_children to any true value (e.g. any non-empty string).

All texts inside the given elements are verified, but possible text outside them is not. By default texts must match exactly, but setting normalize_whitespace to any true value makes text verification independent on newlines, tabs, and the amount of spaces. For more details about handling text see Get Element Text keyword and discussion about elements’ text and tail attributes in the introduction.

Examples using $<XML>$ structure from the introduction:

| ${first} = | Get Element | ${XML} | first |
| Elements Should Be Equal | ${first} | <first id="1">text</first> |
| ${p} = | Get Element | ${XML} | html/p |
| Elements Should Be Equal | ${p} | <p>Text with <b>bold</b> and <i>italics</i>.</p> |
| normalize_whitespace=yes |
| Elements Should Be Equal | ${p} | <p>Text with</p> |

The last example may look a bit strange because the <p> element only has text Text with. The reason is that rest of the text inside <p> actually belongs to the child elements.

See also Elements Should Match.

**elements_should_match**

Verifies that the given source element matches expected.

This keyword works exactly like Elements Should Be Equal except that texts and attribute values in the expected value can be given as patterns.

Pattern matching is similar as matching files in a shell, and it is always case-sensitive. In the pattern, ‘*’ matches anything and ‘?’ matches any single character.

Examples using $<XML>$ structure from the introduction:

| $first = | Get Element | ${XML} | first |
| Elements Should Match | $first | <first id="?">*</first> |

See Elements Should Be Equal for more examples.

**set_element_tag**

Sets the tag of the specified element to tag.

The element whose tag to set is specified using source and xpath. They have exactly the same semantics as with Get Element keyword. The given source structure is modified and also returned.

Examples using $<XML>$ structure from Example:

| Set Element Tag | ${XML} | newTag |
| Should Be Equal | ${XML.tag} | newTag |
| Should Exist | ${XML} | second/child |

New in Robot Framework 2.7.5.

**set_element_text**

Sets text and/or tail text of the specified element.

The element whose text to set is specified using source and xpath. They have exactly the same semantics as with Get Element keyword. The given source structure is modified and also returned.

Element’s text and tail text are changed only if new text and/or tail values are given. See Element attributes section for more information about text and tail in general.

Examples using $<XML>$ structure from Example:

| Set Element Text | ${XML} | new text |
| Should Be | ${XML} | new text |
| Text Should Be | ${XML} | Text with bold&italics. |
| Text Should Be | ${XML} | Text with bold&slanted!! |

New in Robot Framework 2.7.5.
**set_element_attribute** *(source, name, value, xpath='.')*

Sets attribute name of the specified element to value.

The element whose attribute to set is specified using source and xpath. They have exactly the same semantics as with Get Element keyword. The given source structure is modified and also returned.

It is possible to both set new attributes and to overwrite existing. Use Remove Element Attribute or Remove Element Attributes for removing them.

Examples using $\{XML\}$ structure from Example:

| Set Element Attribute | $\{XML\}$ | attr | value |
| Element Attribute Should Be | $\{XML\}$ | id | new | xpath=first |

New in Robot Framework 2.7.5.

**remove_element_attribute** *(source, name, xpath='.')*

Removes attribute name from the specified element.

The element whose attribute to remove is specified using source and xpath. They have exactly the same semantics as with Get Element keyword. The given source structure is modified and also returned.

It is not a failure to remove a non-existing attribute. Use Remove Element Attributes to remove all attributes and Set Element Attribute to set them.

Examples using $\{XML\}$ structure from Example:

| Remove Element Attribute | $\{XML\}$ | id | xpath=first |
| Element Should Not Have Attribute | $\{XML\}$ | id | xpath=first |

New in Robot Framework 2.7.5.

**remove_element_attributes** *(source, xpath=''.)*

Removes all attributes from the specified element.

The element whose attributes to remove is specified using source and xpath. They have exactly the same semantics as with Get Element keyword. The given source structure is modified and also returned.

Use Remove Element Attribute to remove a single attribute and Set Element Attribute to set them.

Examples using $\{XML\}$ structure from Example:

| Remove Element Attributes | $\{XML\}$ | xpath=first |
| Element Should Not Have Attribute | $\{XML\}$ | id | xpath=first |

New in Robot Framework 2.7.5.

**add_element** *(source, element, index=None, xpath=''.)*

Adds a child element to the specified element.

The element to whom to add the new element is specified using source and xpath. They have exactly the same semantics as with Get Element keyword. The given source structure is modified and also returned.

The element to add can be specified as a path to an XML file or as a string containing XML, or it can be an already parsed XML element. The element is copied before adding so modifying either the original or the added element has no effect on the other. The element is added as the last child by default, but a custom index can be used to alter the position. Indices start from zero (0 = first position, 1 = second position, etc.), and negative numbers refer to positions at the end (-1 = second last position, -2 = third last, etc.).

Examples using $\{XML\}$ structure from Example:

| Add Element | $\{XML\}$ | <new id="x">\fäll/</new> |
| Add Element | $\{XML\}$ | <c2/> | xpath=new |
| Index=1 | xpath=new | $\{new\} = | Get Element | $\{XML\}$ | new | Elements Should Be Equal | $\{new\} |

Use Remove Element or Remove Elements to remove elements.

New in Robot Framework 2.7.5.
remove_element (source, xpath='', remove_tail=False)
Removes the element matching xpath from the source structure.

The element to remove from the source is specified with xpath using the same semantics as with Get Element keyword. The given source structure is modified and also returned.

The keyword fails if xpath does not match exactly one element. Use Remove Elements to remove all matched elements.

Element’s tail text is not removed by default, but that can be changed by giving remove_tail a true value (e.g. any non-empty string). See Element attributes section for more information about tail in general.

Examples using $(XML) structure from Example: | Remove Element | $(XML) | xpath=second || Element Should Not Exist | $(XML) | xpath=second || Remove Element | $(XML) | xpath=html/p/b | remove_tail=yes || Element Text Should Be | $(XML) | Text with italics. | xpath=html/p | normalize_whitespace=yes |
New in Robot Framework 2.7.5.

remove_elements (source, xpath='', remove_tail=False)
Removes all elements matching xpath from the source structure.

The elements to remove from the source are specified with xpath using the same semantics as with Get Elements keyword. The given source structure is modified and also returned.

It is not a failure if xpath matches no elements. Use Remove Element to remove exactly one element and Add Element to add new ones.

Element’s tail text is not removed by default, but that can be changed by using remove_tail argument similarly as with Remove Element.

Examples using $(XML) structure from Example: | Remove Elements | $(XML) | xpath=*/child || Element Should Not Exist | $(XML) | xpath=second/child || Element Should Not Exist | $(XML) | xpath=third/child |
New in Robot Framework 2.7.5.

clear_element (source, xpath='.', clear_tail=False)
Clears the contents of the specified element.

The element to clear is specified using source and xpath. They have exactly the same semantics as with Get Element keyword. The given source structure is modified and also returned.

Clearing the element means removing its text, attributes, and children. Element’s tail text is not removed by default, but that can be changed by giving clear_tail a true value (e.g. any non-empty string). See Element attributes section for more information about tail in general.

Examples using $(XML) structure from Example: | Clear Element | $(XML) | xpath=first || Elements Should Be Equal | $(XML) | xpath=first || Elements Should Be Equal | $(XML) | xpath=html/p/b | clear_tail=yes || Element Text Should Be | $(XML) | Text with italics. | xpath=html/p | normalize_whitespace=yes || Clear Element | $(XML) || Elements Should Be Equal | $(XML) | <example/> |
Use Remove Element to remove the whole element.
New in Robot Framework 2.7.5.

copy_element (source, xpath='')
Returns a copy of the specified element.

The element to copy is specified using source and xpath. They have exactly the same semantics as with Get Element keyword.

If the copy or the original element is modified afterwards, the changes have no effect on the other.
Examples using `${XML}` structure from *Example*:  

```
| ${elem} = | Get Element | `${XML}` | xpath=first |
| ${copy1} = | Copy Element | ${elem}  |
| ${copy2} = | Copy Element | `${XML}` | xpath=first |
| Set Element Text | `${XML}` | new text | xpath=first |
| Set Element Attribute | `${copy1}` | id | new |
| Elements Should Be Equal | ${elem} | <first id="1">new text</first> |
| Elements Should Be Equal | ${copy1} | <first id="new">text</first> |
| Elements Should Be Equal | ${copy2} | <first id="1">text</first> |
```

New in Robot Framework 2.7.5.

```python

element_to_string (source, xpath='')
```

Returns the string representation of the specified element.

The element to convert to a string is specified using `source` and `xpath`. They have exactly the same semantics as with `Get Element` keyword.

The returned string is in Unicode format and it does not contain any XML declaration.

See also `Log Element` and `Save XML`.

```python

log_element (source, level='INFO', xpath='')
```

Logs the string representation of the specified element.

The element specified with `source` and `xpath` is first converted into a string using `Element To String` keyword internally. The resulting string is then logged using the given `level`.

The logged string is also returned.

```python

save_xml (source, path, encoding='UTF-8')
```

Saves the given element to the specified file.

The element to save is specified with `source` using the same semantics as with `Get Element` keyword.

The file where the element is saved is denoted with `path` and the encoding to use with `encoding`. The resulting file contains an XML declaration.

Use `Element To String` if you just need a string representation of the element,

New in Robot Framework 2.7.5.

```python

class robot.libraries.XML.ElementComparator (comparator, normalizer=None, exclude_children=False)

    Bases: object

    compare (actual, expected, location=None)

class robot.libraries.XML.Location (path, is_root=True)

    Bases: object

    child (tag)
```

### 4.7.15 `dialogs_ipy` Module

```python

class robot.libraries.dialogs_ipy.MessageDialog (message)

    Bases: robot.libraries.dialogs_ipy._AbstractWinformsDialog

class robot.libraries.dialogs_ipyInputDialog (message, default)

    Bases: robot.libraries.dialogs_ipy._AbstractWinformsDialog

class robot.libraries.dialogs_ipy.SelectionDialog (message, options)

    Bases: robot.libraries.dialogs_ipy._AbstractWinformsDialog

class robot.libraries.dialogs_ipy.PassFailDialog (message)

    Bases: robot.libraries.dialogs_ipy._AbstractWinformsDialog
```
4.7.16 dialogs_jy Module

4.7.17 dialogs_py Module

class robot.libraries.dialogs_py.MessageDialog(message, value=None)
   Bases: robot.libraries.dialogs_py._TkDialog

   after (ms, func=None, *args)
      Call function once after given time.
      MS specifies the time in milliseconds. FUNC gives the function which shall be called. Additional parameters are given as parameters to the function call. Return identifier to cancel scheduling with after_cancel.

   after_cancel (id)
      Cancel scheduling of function identified with ID.
      Identifier returned by after or after_idle must be given as first parameter.

   after_idle (func, *args)
      Call FUNC once if the Tcl main loop has no event to process.
      Return an identifier to cancel the scheduling with after_cancel.

   aspect (minNumer=None, minDenom=None, maxNumer=None, maxDenom=None)
      Instruct the window manager to set the aspect ratio (width/height) of this widget to be between MINNUMBER/MINDENOM and MAXNUMBER/MAXDENOM. Return a tuple of the actual values if no argument is given.

   attributes (*args)
      This subcommand returns or sets platform specific attributes
      The first form returns a list of the platform specific flags and their values. The second form returns the value for the specific option. The third form sets one or more of the values. The values are as follows:
      On Windows, -disabled gets or sets whether the window is in a disabled state. -toolwindow gets or sets the style of the window to toolwindow (as defined in the MSDN). -topmost gets or sets whether this is a topmost window (displays above all other windows).
      On Macintosh, XXXXX
      On Unix, there are currently no special attribute values.

   bbox (column=None, row=None, col2=None, row2=None)
      Return a tuple of integer coordinates for the bounding box of this widget controlled by the geometry manager grid.
      If COLUMN, ROW is given the bounding box applies from the cell with row and column 0 to the specified cell. If COL2 and ROW2 are given the bounding box starts at that cell.
      The returned integers specify the offset of the upper left corner in the master widget and the width and height.

   bell (displayof=0)
      Ring a display’s bell.

   bind (sequence=None, func=None, add=None)
      Bind to this widget at event SEQUENCE a call to function FUNC.
      SEQUENCE is a string of concatenated event patterns. An event pattern is of the form <MODIFIER-MODIFIER-TYPE-Detail> where MODIFIER is one of Control, Mod2, M2, Shift, Mod3, M3, Lock, Mod4, M4, Button1, B1, Mod5, M5 Button2, B2, Meta, M, Button3, B3, Alt, Button4, B4, Double, Button5, B5 Triple, Mod1, M1. TYPE is one of Activate, Enter, Map, ButtonPress, Button, Expose, Motion, ButtonRelease FocusIn, MouseWheel, Circulate, FocusOut, Property, Colormap, Gravity Reparent,
Configure, KeyPress, Key, Unmap, Deactivate, KeyRelease Visibility, Destroy, Leave and DETAIL is the button number for ButtonPress, ButtonRelease and DETAIL is the Keysym for KeyPress and KeyRelease. Examples are <Control-Button-1> for pressing Control and mouse button 1 or <Alt-A> for pressing A and the Alt key (KeyPress can be omitted). An event pattern can also be a virtual event of the form <<AString>> where AString can be arbitrary. This event can be generated by event_generate. If events are concatenated they must appear shortly after each other.

FUNC will be called if the event sequence occurs with an instance of Event as argument. If the return value of FUNC is “break” no further bound function is invoked.

An additional boolean parameter ADD specifies whether FUNC will be called additionally to the other bound function or whether it will replace the previous function.

Bind will return an identifier to allow deletion of the bound function with unbind without memory leak. If FUNC or SEQUENCE is omitted the bound function or list of bound events are returned.

**bind_all**

(sequence=None, func=None, add=None)

Bind to all widgets at an event SEQUENCE a call to function FUNC. An additional boolean parameter ADD specifies whether FUNC will be called additionally to the other bound function or whether it will replace the previous function. See bind for the return value.

**bind_class**

(className, sequence=None, func=None, add=None)

Bind to widgets with bindtag CLASSNAME at event SEQUENCE a call of function FUNC. An additional boolean parameter ADD specifies whether FUNC will be called additionally to the other bound function or whether it will replace the previous function. See bind for the return value.

**bindtags**

(tagList=None)

Set or get the list of bindtags for this widget.

With no argument return the list of all bindtags associated with this widget. With a list of strings as argument the bindtags are set to this list. The bindtags determine in which order events are processed (see bind).

**cget**

(key)

Return the resource value for a KEY given as string.

**client**

(name=None)

Store NAME in WM_CLIENT_MACHINE property of this widget. Return current value.

**clipboard_append**

(string, **kw)

Append STRING to the Tk clipboard.

A widget specified at the optional displayof keyword argument specifies the target display. The clipboard can be retrieved with selection_get.

**clipboard_clear**

(**kw)

Clear the data in the Tk clipboard.

A widget specified for the optional displayof keyword argument specifies the target display.

**clipboard_get**

(**kw)

Retrieve data from the clipboard on window’s display.

The window keyword defaults to the root window of the Tkinter application.

The type keyword specifies the form in which the data is to be returned and should be an atom name such as STRING or FILE_NAME. Type defaults to STRING, except on X11, where the default is to try UTF8_STRING and fall back to STRING.

This command is equivalent to:

selection_get(clipboard)

4.7. libraries Package
colormapwindows (*wlist)
 Store list of window names (WLIST) into WM_COLORMAPWINDOWS property of this widget. This
list contains windows whose colormaps differ from their parents. Return current list of widgets if WLIST
is empty.

colormodel (value=None)
 Useless. Not implemented in Tk.

columnconfigure (index, cnf={}, **kw)
 Configure column INDEX of a grid.

Valid resources are minsize (minimum size of the column), weight (how much does additional space
propagate to this column) and pad (how much space to let additionally).

command (value=None)
 Store VALUE in WM_COMMAND property. It is the command which shall be used to invoke the appli-
cation. Return current command if VALUE is None.

config (cnf=None, **kw)
 Configure resources of a widget.

The values for resources are specified as keyword arguments. To get an overview about the allowed
keyword arguments call the method keys.

configure (cnf=None, **kw)
 Configure resources of a widget.

The values for resources are specified as keyword arguments. To get an overview about the allowed
keyword arguments call the method keys.

deiconify ()
 Deiconify this widget. If it was never mapped it will not be mapped. On Windows it will raise this widget
and give it the focus.

deletecommand (name)
 Internal function.

Delete the Tcl command provided in NAME.

destroy ()
 Destroy this and all descendants widgets.

event_add (virtual, *sequences)
 Bind a virtual event VIRTUAL (of the form <<Name>>) to an event SEQUENCE such that the virtual
event is triggered whenever SEQUENCE occurs.

event_delete (virtual, *sequences)
 Unbind a virtual event VIRTUAL from SEQUENCE.

event_generate (sequence, **kw)
 Generate an event SEQUENCE. Additional keyword arguments specify parameter of the event (e.g. x, y,
rootx, rooty).

event_info (virtual=None)
 Return a list of all virtual events or the information about the SEQUENCE bound to the virtual event
VIRTUAL.

focus ()
 Direct input focus to this widget.

If the application currently does not have the focus this widget will get the focus if the application gets the
focus through the window manager.
focus_displayof()
    Return the widget which has currently the focus on the display where this widget is located.
    Return None if the application does not have the focus.

focus_force()
    Direct input focus to this widget even if the application does not have the focus. Use with caution!

focus_get()
    Return the widget which has currently the focus in the application.
    Use focus_displayof to allow working with several displays. Return None if application does not have the focus.

focus_lastfor()
    Return the widget which would have the focus if top level for this widget gets the focus from the window manager.

focus_set()
    Direct input focus to this widget.
    If the application currently does not have the focus this widget will get the focus if the application gets the focus through the window manager.

focusmodel(model=None)
    Set focus model to MODEL. “active” means that this widget will claim the focus itself, “passive” means that the window manager shall give the focus. Return current focus model if MODEL is None.

frame()
    Return identifier for decorative frame of this widget if present.

geometry(newGeometry=None)
    Set geometry to NEWGEOMETRY of the form =widthxheight+x+y. Return current value if None is given.

getboolean(s)
    Return a boolean value for Tcl boolean values true and false given as parameter.

getdouble
    alias of float

getint
    alias of int

getvar(name='PY_VAR')
    Return value of Tcl variable NAME.

grab_current()
    Return widget which has currently the grab in this application or None.

grab_release()
    Release grab for this widget if currently set.

grab_set()
    Set grab for this widget.
    A grab directs all events to this and descendant widgets in the application.

grab_set_global()
    Set global grab for this widget.
    A global grab directs all events to this and descendant widgets on the display. Use with caution - other applications do not get events anymore.
grab_status()
Return None, “local” or “global” if this widget has no, a local or a global grab.

grid(baseWidth=None, baseHeight=None, widthInc=None, heightInc=None)
Instruct the window manager that this widget shall only be resized on grid boundaries. WIDTHINC and
HEIGHTINC are the width and height of a grid unit in pixels. BASEWIDTH and BASEHEIGHT are the
number of grid units requested in Tk_GeometryRequest.

grid_bbox(column=None, row=None, col2=None, row2=None)
Return a tuple of integer coordinates for the bounding box of this widget controlled by the geometry
manager grid.

If COLUMN, ROW is given the bounding box applies from the cell with row and column 0 to the specified
cell. If COL2 and ROW2 are given the bounding box starts at that cell.

The returned integers specify the offset of the upper left corner in the master widget and the width and
height.

grid_columnconfigure(index, cnf={}, **kw)
Configure column INDEX of a grid.

Valid resources are minsize (minimum size of the column), weight (how much does additional space
propagate to this column) and pad (how much space to let additionally).

grid_location(x, y)
Return a tuple of column and row which identify the cell at which the pixel at position X and Y inside the
master widget is located.

grid_propagate(flag=['_noarg_'])
Set or get the status for propagation of geometry information.

A boolean argument specifies whether the geometry information of the slaves will determine the size of
this widget. If no argument is given, the current setting will be returned.

grid_rowconfigure(index, cnf={}, **kw)
Configure row INDEX of a grid.

Valid resources are minsize (minimum size of the row), weight (how much does additional space propagate
to this row) and pad (how much space to let additionally).

grid_size()
Return a tuple of the number of column and rows in the grid.

grid_slaves(row=None, column=None)
Return a list of all slaves of this widget in its packing order.

group(pathName=None)
Set the group leader widgets for related widgets to PATHNAME. Return the group leader of this widget if
None is given.

iconbitmap(bitmap=None, default=None)
Set bitmap for the iconified widget to BITMAP. Return the bitmap if None is given.

Under Windows, the DEFAULT parameter can be used to set the icon for the widget and any descen-
dents that don’t have an icon set explicitly. DEFAULT can be the relative path to a .ico file (example:
root.iconbitmap(default='myicon.ico') ). See Tk documentation for more information.

iconify()
Display widget as icon.

iconmask(bitmap=None)
Set mask for the icon bitmap of this widget. Return the mask if None is given.
iconname \( (newName=None) \)
Set the name of the icon for this widget. Return the name if None is given.

iconposition \( (x=None, y=None) \)
Set the position of the icon of this widget to X and Y. Return a tuple of the current values of X and X if None is given.

iconwindow \( (pathName=None) \)
Set widget PATHNAME to be displayed instead of icon. Return the current value if None is given.

image_names ()
Return a list of all existing image names.

image_types ()
Return a list of all available image types (e.g. photo bitmap).

keys ()
Return a list of all resource names of this widget.

lift \( (aboveThis=None) \)
Raise this widget in the stacking order.

lower \( (belowThis=None) \)
Lower this widget in the stacking order.

mainloop \( (n=0) \)
Call the mainloop of Tk.

maxsize \( (width=None, height=None) \)
Set max WIDTH and HEIGHT for this widget. If the window is gridded the values are given in grid units. Return the current values if None is given.

minsize \( (width=None, height=None) \)
Set min WIDTH and HEIGHT for this widget. If the window is gridded the values are given in grid units. Return the current values if None is given.

nametowidget \( (name) \)
Return the Tkinter instance of a widget identified by its Tcl name NAME.

option_add \( (pattern, value, priority=None) \)
Set a VALUE (second parameter) for an option PATTERN (first parameter). An optional third parameter gives the numeric priority (defaults to 80).

option_clear ()
Clear the option database.
It will be reloaded if option_add is called.

option_get \( (name, className) \)
Return the value for an option NAME for this widget with CLASSNAME. Values with higher priority override lower values.

option_readfile \( (fileName, priority=None) \)
Read file FILENAME into the option database. An optional second parameter gives the numeric priority.

overrideredirect \( (boolean=None) \)
Instruct the window manager to ignore this widget if BOOLEAN is given with 1. Return the current value if None is given.
pack_propagate (flag=['noarg'])
Set or get the status for propagation of geometry information.
A boolean argument specifies whether the geometry information of the slaves will determine the size of
this widget. If no argument is given the current setting will be returned.

pack_slaves ()
Return a list of all slaves of this widget in its packing order.

place_slaves ()
Return a list of all slaves of this widget in its packing order.

positionfrom (who=None)
Instruct the window manager that the position of this widget shall be defined by the user if WHO is “user”,
and by its own policy if WHO is “program”.

propagate (flag=['noarg'])
Set or get the status for propagation of geometry information.
A boolean argument specifies whether the geometry information of the slaves will determine the size of
this widget. If no argument is given the current setting will be returned.

protocol (name=None, func=None)
Bind function FUNC to command NAME for this widget. Return the function bound to NAME if None is
given. NAME could be e.g. “WM_SAVE_YOURSELF” or “WM_DELETE_WINDOW”.

quit ()
Quit the Tcl interpreter. All widgets will be destroyed.

register (func, subst=None, needcleanup=1)
Return a newly created Tcl function. If this function is called, the Python function FUNC will be executed.
An optional function SUBST can be given which will be executed before FUNC.

resizable (width=None, height=None)
Instruct the window manager whether this width can be resized in WIDTH or HEIGHT. Both values are
boolean values.

rowconfigure (index, cnf={}, **kw)
Configure row INDEX of a grid.
Valid resources are minsize (minimum size of the row), weight (how much does additional space propagate
to this row) and pad (how much space to let additionally).

selection_clear (**kw)
Clear the current X selection.

selection_get (**kw)
Return the contents of the current X selection.
A keyword parameter selection specifies the name of the selection and defaults to PRIMARY. A keyword
parameter displayof specifies a widget on the display to use. A keyword parameter type specifies the
form of data to be fetched, defaulting to STRING except on X11, where UTF8_STRING is tried before
STRING.

selection_handle (command, **kw)
Specify a function COMMAND to call if the X selection owned by this widget is queried by another
application.
This function must return the contents of the selection. The function will be called with the arguments
OFFSET and LENGTH which allows the chunking of very long selections. The following keyword pa-
rameters can be provided: selection - name of the selection (default PRIMARY), type - type of the selection
(e.g. STRING, FILE_NAME).
selection_owner(**kw)
   Become owner of X selection.
   A keyword parameter selection specifies the name of the selection (default PRIMARY).

selection_owner_get(**kw)
   Return owner of X selection.
   The following keyword parameter can be provided: selection - name of the selection (default PRIMARY),
   type - type of the selection (e.g. STRING, FILE_NAME).

send(interp, cmd, *args)
   Send Tcl command CMD to different interpreter INTERP to be executed.

setvar(name='PY_VAR', value='1')
   Set Tcl variable NAME to VALUE.

show()

size()
   Return a tuple of the number of column and rows in the grid.

sizefrom(who=None)
   Instruct the window manager that the size of this widget shall be defined by the user if WHO is “user”, and
   by its own policy if WHO is “program”.

slaves()
   Return a list of all slaves of this widget in its packing order.

state(newstate=None)
   Query or set the state of this widget as one of normal, icon, iconic (see wm_iconwindow), withdrawn, or
   zoomed (Windows only).

title(string=None)
   Set the title of this widget.

tk_bisque()
   Change the color scheme to light brown as used in Tk 3.6 and before.

tk_focusFollowsMouse()
   The widget under mouse will get automatically focus. Can not be disabled easily.

tk_focusNext()
   Return the next widget in the focus order which follows widget which has currently the focus.
   The focus order first goes to the next child, then to the children of the child recursively and then to the next
   sibling which is higher in the stacking order. A widget is omitted if it has the takefocus resource set to 0.

tk_focusPrev()
   Return previous widget in the focus order. See tk_focusNext for details.

tk_menuBar(*args)
   Do not use. Needed in Tk 3.6 and earlier.

tk_setPalette(*args, **kw)
   Set a new color scheme for all widget elements.
   A single color as argument will cause that all colors of Tk widget elements are derived from this. Alternat-
   ively several keyword parameters and its associated colors can be given. The following keywords are
   valid: activeBackground, foreground, selectColor, activeForeground, highlightBackground, selectBack-
   ground, background, highlightColor, selectForeground, disabledForeground, insertBackground, trough-
   Color.
tk_strictMotif(boolean=None)
    Set Tcl internal variable, whether the look and feel should adhere to Motif.
    A parameter of 1 means adhere to Motif (e.g. no color change if mouse passes over slider). Returns the
    set value.

tkraise(aboveThis=None)
    Raise this widget in the stacking order.

transient(master=None)
    Instruct the window manager that this widget is transient with regard to widget MASTER.

unbind(sequence, funcid=None)
    Unbind for this widget for event SEQUENCE the function identified with FUNCID.

unbind_all(sequence)
    Unbind for all widgets for event SEQUENCE all functions.

unbind_class(className, sequence)
    Unbind for all widgets with bindtag CLASSNAME for event SEQUENCE all functions.

update()
    Enter event loop until all pending events have been processed by Tcl.

update_idletasks()
    Enter event loop until all idle callbacks have been called. This will update the display of windows but not
    process events caused by the user.

wait_variable(name='PY_VAR')
    Wait until the variable is modified.
    A parameter of type IntVar, StringVar, DoubleVar or BooleanVar must be given.

wait_visibility(window=None)
    Wait until the visibility of a WIDGET changes (e.g. it appears).
    If no parameter is given self is used.

wait_window(window=None)
    Wait until a WIDGET is destroyed.
    If no parameter is given self is used.

waitvar(name='PY_VAR')
    Wait until the variable is modified.
    A parameter of type IntVar, StringVar, DoubleVar or BooleanVar must be given.

winfo_atom(name, displayof=0)
    Return integer which represents atom NAME.

winfo_atomname(id, displayof=0)
    Return name of atom with identifier ID.

winfo_cells()
    Return number of cells in the colormap for this widget.

winfo_children()
    Return a list of all widgets which are children of this widget.

winfo_class()
    Return window class name of this widget.

winfo_colormapfull()
    Return true if at the last color request the colormap was full.
**winfo_containing** (*rootX, rootY, displayof=0*)

Return the widget which is at the root coordinates ROOTX, ROOTY.

**winfo_depth**()

Return the number of bits per pixel.

**winfo_exists**()

Return true if this widget exists.

**winfo_fpixels** (*number*)

Return the number of pixels for the given distance NUMBER (e.g. “3c”) as float.

**winfo_geometry**()

Return geometry string for this widget in the form “widthxheight+X+Y”.

**winfo_height**()

Return height of this widget.

**winfo_id**()

Return identifier ID for this widget.

**winfo_interps** (*displayof=0*)

Return the name of all Tcl interpreters for this display.

**winfo_ismapped**()

Return true if this widget is mapped.

**winfo_manager**()

Return the window manager name for this widget.

**winfo_name**()

Return the name of this widget.

**winfo_parent**()

Return the name of the parent of this widget.

**winfo_pathname** (*id, displayof=0*)

Return the pathname of the widget given by ID.

**winfo_pixels** (*number*)

Rounded integer value of winfo_fpixels.

**winfo_pointerx**()

Return the x coordinate of the pointer on the root window.

**winfo_pointerxy**()

Return a tuple of x and y coordinates of the pointer on the root window.

**winfo_pointery**()

Return the y coordinate of the pointer on the root window.

**winfo_reqheight**()

Return requested height of this widget.

**winfo_reqwidth**()

Return requested width of this widget.

**winfo_rgb** (*color*)

Return tuple of decimal values for red, green, blue for COLOR in this widget.

**winfo_rootx**()

Return x coordinate of upper left corner of this widget on the root window.
wininfo_rooty()
    Return y coordinate of upper left corner of this widget on the root window.

wininfo_screen()
    Return the screen name of this widget.

wininfo_screencells()
    Return the number of the cells in the colormap of the screen of this widget.

wininfo_screendepth()
    Return the number of bits per pixel of the root window of the screen of this widget.

wininfo_screendepth()
    Return the number of pixels of the height of the screen of this widget in pixel.

wininfo_screenmmheight()
    Return the number of pixels of the height of the screen of this widget in mm.

wininfo_screenmmwidth()
    Return the number of pixels of the width of the screen of this widget in mm.

wininfo_screenvisual()
    Return one of the strings directcolor, grayscale, pseudocolor, staticcolor, staticgray, or truecolor for the
default colormodel of this screen.

wininfo_screenwidth()
    Return the number of pixels of the width of the screen of this widget in pixel.

wininfo_server()
    Return information of the X-Server of the screen of this widget in the form “XmajorRminor vendor vendorVersion”.

wininfo_toplevel()
    Return the toplevel widget of this widget.

wininfo_viewable()
    Return true if the widget and all its higher ancestors are mapped.

wininfo_visual()
    Return one of the strings directcolor, grayscale, pseudocolor, staticcolor, staticgray, or truecolor for the
colormodel of this widget.

wininfo_visualid()
    Return the X identifier for the visual for this widget.

wininfo_visualsavailable(includeids=0)
    Return a list of all visuals available for the screen of this widget.
    Each item in the list consists of a visual name (see wininfo_visual), a depth and if INCLUDEIDS=1 is given
    also the X identifier.

wininfo_vrootheight()
    Return the height of the virtual root window associated with this widget in pixels. If there is no virtual root
    window return the height of the screen.

wininfo_vrootwidth()
    Return the width of the virtual root window associated with this widget in pixel. If there is no virtual root
    window return the width of the screen.

wininfo_vrootx()
    Return the x offset of the virtual root relative to the root window of the screen of this widget.
winfo_vrooty()
  Return the y offset of the virtual root relative to the root window of the screen of this widget.

winfo_width()
  Return the width of this widget.

winfo_x()
  Return the x coordinate of the upper left corner of this widget in the parent.

winfo_y()
  Return the y coordinate of the upper left corner of this widget in the parent.

withdraw()
  Withdraw this widget from the screen such that it is unmapped and forgotten by the window manager. Re-draw it with wm_deiconify.

wm_aspect(minNumer=None, minDenom=None, maxNumer=None, maxDenom=None)
  Instruct the window manager to set the aspect ratio (width/height) of this widget to be between MINNU-MER/MINDENOM and MAXNUMBER/MAXDENOM. Return a tuple of the actual values if no argument is given.

wm_attributes(*args)
  This subcommand returns or sets platform specific attributes
  The first form returns a list of the platform specific flags and their values. The second form returns the value for the specific option. The third form sets one or more of the values. The values are as follows:
  On Windows, -disabled gets or sets whether the window is in a disabled state. -toolwindow gets or sets the style of the window to toolwindow (as defined in the MSDN). -topmost gets or sets whether this is a topmost window (displays above all other windows).
  On Macintosh, XXXXX
  On Unix, there are currently no special attribute values.

wm_client(name=None)
  Store NAME in WM_CLIENT_MACHINE property of this widget. Return current value.

wm_colormapwindows(*wlist)
  Store list of window names (WLIST) into WM_COLORMAPWINDOWS property of this widget. This list contains windows whose colormaps differ from their parents. Return current list of widgets if WLIST is empty.

wm_command(value=None)
  Store VALUE in WM_COMMAND property. It is the command which shall be used to invoke the application. Return current command if VALUE is None.

wm_deiconify()
  Deiconify this widget. If it was never mapped it will not be mapped. On Windows it will raise this widget and give it the focus.

wm_focusmodel(model=None)
  Set focus model to MODEL. “active” means that this widget will claim the focus itself, “passive” means that the window manager shall give the focus. Return current focus model if MODEL is None.

wm_frame()
  Return identifier for decorative frame of this widget if present.

wm_geometry(newGeometry=None)
  Set geometry to NEWGEOMETRY of the form =widthxheight+x+y. Return current value if None is given.

wm_grid(baseWidth=None, baseHeight=None, widthInc=None, heightInc=None)
  Instruct the window manager that this widget shall only be resized on grid boundaries. WIDTHINC and
HEIGHTINC are the width and height of a grid unit in pixels. BASEWIDTH and BASEHEIGHT are the number of grid units requested in Tk_GeometryRequest.

wm_group (pathName=None)
Set the group leader widgets for related widgets to PATHNAME. Return the group leader of this widget if None is given.

wm_iconbitmap (bitmap=None, default=None)
Set bitmap for the iconified widget to BITMAP. Return the bitmap if None is given.
Under Windows, the DEFAULT parameter can be used to set the icon for the widget and any descendants that don’t have an icon set explicitly. DEFAULT can be the relative path to a .ico file (example: root.iconbitmap(default=’myicon.ico’)). See Tk documentation for more information.

wm_iconify ()
Display widget as icon.

wm_iconmask (bitmap=None)
Set mask for the icon bitmap of this widget. Return the mask if None is given.

wm_iconname (newName=None)
Set the name of the icon for this widget. Return the name if None is given.

wm_iconposition (x=None, y=None)
Set the position of the icon of this widget to X and Y. Return a tuple of the current values of X and X if None is given.

wm_iconwindow (pathName=None)
Set widget PATHNAME to be displayed instead of icon. Return the current value if None is given.

wm_maxsize (width=None, height=None)
Set max WIDTH and HEIGHT for this widget. If the window is gridded the values are given in grid units. Return the current values if None is given.

wm_minsize (width=None, height=None)
Set min WIDTH and HEIGHT for this widget. If the window is gridded the values are given in grid units. Return the current values if None is given.

wm_overrideredirect (boolean=None)
Instruct the window manager to ignore this widget if BOOLEAN is given with 1. Return the current value if None is given.

wm_positionfrom (who=None)
Instruct the window manager that the position of this widget shall be defined by the user if WHO is “user”, and by its own policy if WHO is “program”.

wm_protocol (name=None, func=None)
Bind function FUNC to command NAME for this widget. Return the function bound to NAME if None is given. NAME could be e.g. “WM_SAVE_YOURSELF” or “WM_DELETE_WINDOW”.

wm_resizable (width=None, height=None)
Instruct the window manager whether this width can be resized in WIDTH or HEIGHT. Both values are boolean values.

wm_sizefrom (who=None)
Instruct the window manager that the size of this widget shall be defined by the user if WHO is “user”, and by its own policy if WHO is “program”.

wm_state (newstate=None)
Query or set the state of this widget as one of normal, icon, iconic (see wm_iconwindow), withdrawn, or zoomed (Windows only).
**wm_title**(string=None)
Set the title of this widget.

**wm_transient**(master=None)
Instruct the window manager that this widget is transient with regard to widget MASTER.

**wm_withdraw()**
Withdraw this widget from the screen such that it is unmapped and forgotten by the window manager. Re-draw it with `wm_deiconify`.

**class** `robot.libraries.dialogs_py.InputDialog**(message, default='''
Bases: robot.libraries.dialogs_py._TkDialog

**after**(ms, func=None, *args)
Call function once after given time.

MS specifies the time in milliseconds. FUNC gives the function which shall be called. Additional parameters are given as parameters to the function call. Return identifier to cancel scheduling with `after_cancel`.

**after_cancel**(id)
Cancel scheduling of function identified with ID.

Identifier returned by after or after_idle must be given as first parameter.

**after_idle**(func, *args)
Call FUNC once if the Tcl main loop has no event to process.

Return an identifier to cancel the scheduling with `after_cancel`.

**aspect**(minNumer=None, minDenom=None, maxNumer=None, maxDenom=None)
Instruct the window manager to set the aspect ratio (width/height) of this widget to be between MINNU-MER/MINDENOM and MAXNUMER/MAXDENOM. Return a tuple of the actual values if no argument is given.

**attributes**(args)
This subcommand returns or sets platform specific attributes

The first form returns a list of the platform specific flags and their values. The second form returns the value for the specific option. The third form sets one or more of the values. The values are as follows:

On Windows, -disabled gets or sets whether the window is in a disabled state. -toolwindow gets or sets the style of the window to toolwindow (as defined in the MSDN). -topmost gets or sets whether this is a topmost window (displays above all other windows).

On Macintosh, XXXXX

On Unix, there are currently no special attribute values.

**bbox**(column=None, row=None, col2=None, row2=None)
Return a tuple of integer coordinates for the bounding box of this widget controlled by the geometry manager grid.

If COLUMN, ROW is given the bounding box applies from the cell with row and column 0 to the specified cell. If COL2 and ROW2 are given the bounding box starts at that cell.

The returned integers specify the offset of the upper left corner in the master widget and the width and height.

**bell**(displayof=0)
Ring a display’s bell.

**bind**(sequence=None, func=None, add=None)
Bind to this widget at event SEQUENCE a call to function FUNC.
SEQUENCE is a string of concatenated event patterns. An event pattern is of the form `<MODIFIER-MODIFIER-TYPE-DETAIL>` where MODIFIER is one of Control, Mod2, M2, Shift, Mod3, M3, Lock, Mod4, M4, Button1, B1, Mod5, M5 Button2, B2, Meta, M, Button3, B3, Alt, Button4, B4, Double, Button5, B5 Triple, Mod1, M1. TYPE is one of Activate, Enter, Map, ButtonPress, Button, Expose, Motion, ButtonRelease FocusIn, MouseWheel, Circulate, FocusOut, Property, Colormap, Gravity Reparent, Configure, KeyPress, Key, Unmap, Deactivate, KeyRelease Visibility, Destroy, Leave and DETAIL is the button number for ButtonPress, ButtonRelease and DETAIL is the Keysym for KeyPress and KeyRelease. Examples are `<Control-Button-1>` for pressing Control and mouse button 1 or `<Alt-A>` for pressing A and the Alt key (KeyPress can be omitted). An event pattern can also be a virtual event of the form `<<<AString>>` where AString can be arbitrary. This event can be generated by `event_generate`. If events are concatenated they must appear shortly after each other.

FUNC will be called if the event sequence occurs with an instance of `Event` as argument. If the return value of FUNC is “break” no further bound function is invoked.

An additional boolean parameter ADD specifies whether FUNC will be called additionally to the other bound function or whether it will replace the previous function.

Bind will return an identifier to allow deletion of the bound function with unbind without memory leak.

If FUNC or SEQUENCE is omitted the bound function or list of bound events are returned.

```
bind_all (sequence=None, func=None, add=None)
```

Bind to all widgets at an event SEQUENCE a call to function FUNC. An additional boolean parameter ADD specifies whether FUNC will be called additionally to the other bound function or whether it will replace the previous function. See bind for the return value.

```
bind_class (className, sequence=None, func=None, add=None)
```

Bind to widgets with bindtag CLASSNAME at event SEQUENCE a call of function FUNC. An additional boolean parameter ADD specifies whether FUNC will be called additionally to the other bound function or whether it will replace the previous function. See bind for the return value.

```
bindtags (tagList=None)
```

Set or get the list of bindtags for this widget.

With no argument return the list of all bindtags associated with this widget. With a list of strings as argument the bindtags are set to this list. The bindtags determine in which order events are processed (see bind).

```
cget (key)
```

Return the resource value for a KEY given as string.

```
client (name=None)
```

Store NAME in WM_CLIENT_MACHINE property of this widget. Return current value.

```
clipboard_append (string, **kw)
```

Append STRING to the Tk clipboard.

A widget specified at the optional displayof keyword argument specifies the target display. The clipboard can be retrieved with `selection_get`.

```
clipboard_clear (**kw)
```

Clear the data in the Tk clipboard.

A widget specified for the optional displayof keyword argument specifies the target display.

```
clipboard_get (**kw)
```

Retrieve data from the clipboard on window’s display.

The window keyword defaults to the root window of the Tkinter application.
The type keyword specifies the form in which the data is to be returned and should be an atom name such as STRING or FILE_NAME. Type defaults to STRING, except on X11, where the default is to try UTF8_STRING and fall back to STRING.

This command is equivalent to:

```
selection_get(CLIPBOARD)
```

**colormapwindows (*wlist)**

Store list of window names (WLIST) into WM_COLORMAPWINDOWS property of this widget. This list contains windows whose colormaps differ from their parents. Return current list of widgets if WLIST is empty.

**colormodel (value=None)**

Useless. Not implemented in Tk.

**columnconfigure (index, cnf={}, **kw)**

Configure column INDEX of a grid.

Valid resources are minsize (minimum size of the column), weight (how much does additional space propagate to this column) and pad (how much space to let additionally).

**command (value=None)**

Store VALUE in WM_COMMAND property. It is the command which shall be used to invoke the application. Return current command if VALUE is None.

**config (cnf=None, **kw)**

Configure resources of a widget.

The values for resources are specified as keyword arguments. To get an overview about the allowed keyword arguments call the method keys.

**configure (cnf=None, **kw)**

Configure resources of a widget.

The values for resources are specified as keyword arguments. To get an overview about the allowed keyword arguments call the method keys.

**deiconify()**

Deiconify this widget. If it was never mapped it will not be mapped. On Windows it will raise this widget and give it the focus.

**deletecommand (name)**

Internal function.

Delete the Tcl command provided in NAME.

**destroy()**

Destroy this and all descendants widgets.

**event_add (virtual, *sequences)**

Bind a virtual event VIRTUAL (of the form <<Name>>) to an event SEQUENCE such that the virtual event is triggered whenever SEQUENCE occurs.

**event_delete (virtual, *sequences)**

Unbind a virtual event VIRTUAL from SEQUENCE.

**event_generate (sequence, **kw)**

Generate an event SEQUENCE. Additional keyword arguments specify parameter of the event (e.g. x, y, rootx, rooty).
event_info(virtual=None)
   Return a list of all virtual events or the information about the SEQUENCE bound to the virtual event VIRTUAL.

focus()
   Direct input focus to this widget.
   If the application currently does not have the focus this widget will get the focus if the application gets the focus through the window manager.

focus_displayof()
   Return the widget which has currently the focus on the display where this widget is located.
   Return None if the application does not have the focus.

focus_force()
   Direct input focus to this widget even if the application does not have the focus. Use with caution!

focus_get()
   Return the widget which has currently the focus in the application.
   Use focus_displayof to allow working with several displays. Return None if application does not have the focus.

focus_lastfor()
   Return the widget which would have the focus if top level for this widget gets the focus from the window manager.

focus_set()
   Direct input focus to this widget.
   If the application currently does not have the focus this widget will get the focus if the application gets the focus through the window manager.

focusmodel(model=None)
   Set focus model to MODEL. “active” means that this widget will claim the focus itself, “passive” means that the window manager shall give the focus. Return current focus model if MODEL is None.

frame()
   Return identifier for decorative frame of this widget if present.

gameometry(newGeometry=None)
   Set geometry to NEWGEOMETRY of the form =widthxheight+x+y. Return current value if None is given.

getboolean(s)
   Return a boolean value for Tcl boolean values true and false given as parameter.

getdouble
   alias of float

getint
   alias of int

getvar(name='PY_VAR')
   Return value of Tcl variable NAME.

grab_current()
   Return widget which has currently the grab in this application or None.

grab_release()
   Release grab for this widget if currently set.

grab_set()
   Set grab for this widget.
A grab directs all events to this and descendant widgets in the application.

```robot
grab_set_global()
```

Set global grab for this widget.

A global grab directs all events to this and descendant widgets on the display. Use with caution - other applications do not get events anymore.

```robot
grab_status()
```

Return None, “local” or “global” if this widget has no, a local or a global grab.

```robot
grid(baseWidth=None, baseHeight=None, widthInc=None, heightInc=None)
```

Instruct the window manager that this widget shall only be resized on grid boundaries. WIDTHINC and HEIGHTINC are the width and height of a grid unit in pixels. BASEWIDTH and BASEHEIGHT are the number of grid units requested in Tk_GeometryRequest.

```robot
grid_bbox(column=None, row=None, col2=None, row2=None)
```

Return a tuple of integer coordinates for the bounding box of this widget controlled by the geometry manager grid.

If COLUMN, ROW is given the bounding box applies from the cell with row and column 0 to the specified cell. If COL2 and ROW2 are given the bounding box starts at that cell.

The returned integers specify the offset of the upper left corner in the master widget and the width and height.

```robot
grid_columnconfigure(index, cnf={}, **kw)
```

Configure column INDEX of a grid.

Valid resources are minsize (minimum size of the column), weight (how much does additional space propagate to this column) and pad (how much space to let additionally).

```robot
grid_location(x, y)
```

Return a tuple of column and row which identify the cell at which the pixel at position X and Y inside the master widget is located.

```robot
grid_propagate(flag=['_noarg_'])
```

Set or get the status for propagation of geometry information.

A boolean argument specifies whether the geometry information of the slaves will determine the size of this widget. If no argument is given, the current setting will be returned.

```robot
grid_rowconfigure(index, cnf={}, **kw)
```

Configure row INDEX of a grid.

Valid resources are minsize (minimum size of the row), weight (how much does additional space propagate to this row) and pad (how much space to let additionally).

```robot
grid_size()
```

Return a tuple of the number of column and rows in the grid.

```robot
grid_slaves(row=None, column=None)
```

Return a list of all slaves of this widget in its packing order.

```robot
group(pathName=None)
```

Set the group leader widgets for related widgets to PATHNAME. Return the group leader of this widget if None is given.

```robot
iconbitmap(bitmap=None, default=None)
```

Set bitmap for the iconified widget to BITMAP. Return the bitmap if None is given.
Under Windows, the DEFAULT parameter can be used to set the icon for the widget and any descendants that don’t have an icon set explicitly. DEFAULT can be the relative path to a .ico file (example: root.iconbitmap(default=’myicon.ico’ )). See Tk documentation for more information.

```python
iconify()
    Display widget as icon.

iconmask(bitmap=None)
    Set mask for the icon bitmap of this widget. Return the mask if None is given.

iconname(newName=None)
    Set the name of the icon for this widget. Return the name if None is given.

iconposition(x=None, y=None)
    Set the position of the icon of this widget to X and Y. Return a tuple of the current values of X and X if None is given.

iconwindow(pathName=None)
    Set widget PATHNAME to be displayed instead of icon. Return the current value if None is given.

image_names()
    Return a list of all existing image names.

image_types()
    Return a list of all available image types (e.g. photo bitmap).

keys()
    Return a list of all resource names of this widget.

lift(aboveThis=None)
    Raise this widget in the stacking order.

lower(belowThis=None)
    Lower this widget in the stacking order.

mainloop(n=0)
    Call the mainloop of Tk.

maxsize(width=None, height=None)
    Set max WIDTH and HEIGHT for this widget. If the window is gridded the values are given in grid units. Return the current values if None is given.

minsize(width=None, height=None)
    Set min WIDTH and HEIGHT for this widget. If the window is gridded the values are given in grid units. Return the current values if None is given.

nametowidget(name)
    Return the Tkinter instance of a widget identified by its Tcl name NAME.

option_add(pattern, value, priority=None)
    Set a VALUE (second parameter) for an option PATTERN (first parameter). An optional third parameter gives the numeric priority (defaults to 80).

option_clear()
    Clear the option database.
    It will be reloaded if option_add is called.

option_get(name, className)
    Return the value for an option NAME for this widget with CLASSNAME. Values with higher priority override lower values.
option_readfile (fileName, priority=None)
    Read file FILENAME into the option database.
    An optional second parameter gives the numeric priority.

overrideredirect (boolean=None)
    Instruct the window manager to ignore this widget if BOOLEAN is given with 1. Return the current value
    if None is given.

pack_propagate (flag=['_noarg_'])
    Set or get the status for propagation of geometry information.
    A boolean argument specifies whether the geometry information of the slaves will determine the size of
    this widget. If no argument is given the current setting will be returned.

pack_slaves ()
    Return a list of all slaves of this widget in its packing order.

place_slaves ()
    Return a list of all slaves of this widget in its packing order.

positionfrom (who=None)
    Instruct the window manager that the position of this widget shall be defined by the user if WHO is “user”,
    and by its own policy if WHO is “program”.

propagate (flag=['_noarg_'])
    Set or get the status for propagation of geometry information.
    A boolean argument specifies whether the geometry information of the slaves will determine the size of
    this widget. If no argument is given the current setting will be returned.

protocol (name=None, func=None)
    Bind function FUNC to command NAME for this widget. Return the function bound to NAME if None is
    given. NAME could be e.g. “WM_SAVE_YOURSELF” or “WM_DELETE_WINDOW”.

quit ()
    Quit the Tcl interpreter. All widgets will be destroyed.

register (func, subst=None, needcleanup=1)
    Return a newly created Tcl function. If this function is called, the Python function FUNC will be executed.
    An optional function SUBST can be given which will be executed before FUNC.

resizable (width=None, height=None)
    Instruct the window manager whether this width can be resized in WIDTH or HEIGHT. Both values are
    boolean values.

rowconfigure (index, cnf={}, **kw)
    Configure row INDEX of a grid.
    Valid resources are minsize (minimum size of the row), weight (how much does additional space propagate
to this row) and pad (how much space to let additionally).

selection_clear (**kw)
    Clear the current X selection.

selection_get (**kw)
    Return the contents of the current X selection.
    A keyword parameter selection specifies the name of the selection and defaults to PRIMARY. A keyword
    parameter displayof specifies a widget on the display to use. A keyword parameter type specifies the
    form of data to be fetched, defaulting to STRING except on X11, where UTF8_STRING is tried before
    STRING.
**selection_handle** *(command, **kw)*

Specify a function COMMAND to call if the X selection owned by this widget is queried by another application.

This function must return the contents of the selection. The function will be called with the arguments OFFSET and LENGTH which allows the chunking of very long selections. The following keyword parameters can be provided: selection - name of the selection (default PRIMARY), type - type of the selection (e.g. STRING, FILE_NAME).

**selection_own** (**kw**)

Become owner of X selection.

A keyword parameter selection specifies the name of the selection (default PRIMARY).

**selection_own_get** (**kw**)

Return owner of X selection.

The following keyword parameter can be provided: selection - name of the selection (default PRIMARY), type - type of the selection (e.g. STRING, FILE_NAME).

**send** *(interp, cmd, *args)*

Send Tcl command CMD to different interpreter INTERP to be executed.

**setvar** *(name='PY_VAR', value='1')*

Set Tcl variable NAME to VALUE.

**show**

**size**

Return a tuple of the number of column and rows in the grid.

**sizefrom** *(who=None)*

Instruct the window manager that the size of this widget shall be defined by the user if WHO is “user”, and by its own policy if WHO is “program”.

**slaves**

Return a list of all slaves of this widget in its packing order.

**state** *(newstate=None)*

Query or set the state of this widget as one of normal, icon, iconic (see wm_iconwindow), withdrawn, or zoomed (Windows only).

**title** *(string=None)*

Set the title of this widget.

**tk_bisque**

Change the color scheme to light brown as used in Tk 3.6 and before.

**tk_focusFollowsMouse**

The widget under mouse will get automatically focus. Can not be disabled easily.

**tk_focusNext**

Return the next widget in the focus order which follows widget which has currently the focus.

The focus order first goes to the next child, then to the children of the child recursively and then to the next sibling which is higher in the stacking order. A widget is omitted if it has the takefocus resource set to 0.

**tk_focusPrev**

Return previous widget in the focus order. See tk_focusNext for details.

**tk_menuBar** *(*args)*

Do not use. Needed in Tk 3.6 and earlier.
**tk_setPalette** (*args, **kw*)
Set a new color scheme for all widget elements.

A single color as argument will cause that all colors of Tk widget elements are derived from this. Alternatively several keyword parameters and its associated colors can be given. The following keywords are valid: activeBackground, foreground, selectColor, activeForeground, highlightBackground, selectBackground, background, highlightColor, selectForeground, disabledForeground, insertBackground, troughColor.

**tk_strictMotif** (boolean=None)
Set Tcl internal variable, whether the look and feel should adhere to Motif.

A parameter of 1 means adhere to Motif (e.g. no color change if mouse passes over slider). Returns the set value.

**tkraise** (aboveThis=None)
Raise this widget in the stacking order.

**transient** (master=None)
Instruct the window manager that this widget is transient with regard to widget MASTER.

**unbind** (sequence, funcid=None)
Unbind for this widget for event SEQUENCE the function identified with FUNCID.

**unbind_all** (sequence)
Unbind for all widgets for event SEQUENCE all functions.

**unbind_class** (className, sequence)
Unbind for a all widgets with bindtag CLASSNAME for event SEQUENCE all functions.

**update** ()
Enter event loop until all pending events have been processed by Tcl.

**update_idletasks** ()
Enter event loop until all idle callbacks have been called. This will update the display of windows but not process events caused by the user.

**wait_variable** (name='PY_VAR')
Wait until the variable is modified.

A parameter of type IntVar, StringVar, DoubleVar or BooleanVar must be given.

**wait_visibility** (window=None)
Wait until the visibility of a WIDGET changes (e.g. it appears).

If no parameter is given self is used.

**wait_window** (window=None)
Wait until a WIDGET is destroyed.

If no parameter is given self is used.

**waitvar** (name='PY_VAR')
Wait until the variable is modified.

A parameter of type IntVar, StringVar, DoubleVar or BooleanVar must be given.

**winfo_atom** (name, displayof=0)
Return integer which represents atom NAME.

**winfo_atomname** (id, displayof=0)
Return name of atom with identifier ID.
wininfo_cells()
Return number of cells in the colormap for this widget.

wininfo_children()
Return a list of all widgets which are children of this widget.

wininfo_class()
Return window class name of this widget.

wininfo_colormapfull()
Return true if at the last color request the colormap was full.

wininfo_containing(rootX, rootY, displayof=0)
Return the widget which is at the root coordinates ROOTX, ROOTY.

wininfo_depth()
Return the number of bits per pixel.

wininfo_exists()
Return true if this widget exists.

wininfo_fpixels(number)
Return the number of pixels for the given distance NUMBER (e.g. “3c”) as float.

wininfo_geometry()
Return geometry string for this widget in the form “widthxheight+X+Y”.

wininfo_height()
Return height of this widget.

wininfo_id()
Return identifier ID for this widget.

wininfo_interps(displayof=0)
Return the name of all Tcl interpreters for this display.

wininfo_ismapped()
Return true if this widget is mapped.

wininfo_manager()
Return the window mananger name for this widget.

wininfo_name()
Return the name of this widget.

wininfo_parent()
Return the name of the parent of this widget.

wininfo_pathname(id, displayof=0)
Return the pathname of the widget given by ID.

wininfo_pixels(number)
Rounded integer value of wininfo_fpixels.

wininfo_pointerx()
Return the x coordinate of the pointer on the root window.

wininfo_pointerxy()
Return a tuple of x and y coordinates of the pointer on the root window.

wininfo_pointery()
Return the y coordinate of the pointer on the root window.
\texttt{winfo\_reqheight()}  
Return requested height of this widget.

\texttt{winfo\_reqwidth()}  
Return requested width of this widget.

\texttt{winfo\_rgb}(\textit{color})  
Return tuple of decimal values for red, green, blue for \textit{COLOR} in this widget.

\texttt{winfo\_rootx()}  
Return x coordinate of upper left corner of this widget on the root window.

\texttt{winfo\_rooty()}  
Return y coordinate of upper left corner of this widget on the root window.

\texttt{winfo\_screen()}  
Return the screen name of this widget.

\texttt{winfo\_screencells()}  
Return the number of the cells in the colormap of the screen of this widget.

\texttt{winfo\_screendepth()}  
Return the number of bits per pixel of the root window of the screen of this widget.

\texttt{winfo\_screenheight()}  
Return the number of pixels of the height of the screen of this widget in pixel.

\texttt{winfo\_screenmmheight()}  
Return the number of pixels of the height of the screen of this widget in mm.

\texttt{winfo\_screenwidth()}  
Return the number of pixels of the width of the screen of this widget in pixel.

\texttt{winfo\_screenwidth()}  
Return the number of pixels of the width of the screen of this widget in mm.

\texttt{winfo\_screenwidth()}  
Return one of the strings directcolor, grayscale, pseudocolor, staticcolor, staticgray, or truecolor for the default colormodel of this screen.

\texttt{winfo\_screenwidth()}  
Return the number of pixels of the width of the screen of this widget in pixel.

\texttt{winfo\_server()}  
Return information of the X-Server of the screen of this widget in the form “XmajorRminor vendor vendorVersion”.

\texttt{winfo\_toplevel()}  
Return the toplevel widget of this widget.

\texttt{winfo\_viewable()}  
Return true if the widget and all its higher ancestors are mapped.

\texttt{winfo\_visual()}  
Return one of the strings directcolor, grayscale, pseudocolor, staticcolor, staticgray, or truecolor for the colormodel of this widget.

\texttt{winfo\_visualid()}  
Return the X identifier for the visual for this widget.

\texttt{winfo\_visualsavailable}(\textit{includeids}=0)  
Return a list of all visuals available for the screen of this widget.

Each item in the list consists of a visual name (see \texttt{winfo\_visual}), a depth and if \texttt{INCLUDEIDS}=1 is given also the X identifier.
winfo_vrootheight ()
Return the height of the virtual root window associated with this widget in pixels. If there is no virtual root window return the height of the screen.

winfo_vrootwidth ()
Return the width of the virtual root window associated with this widget in pixel. If there is no virtual root window return the width of the screen.

winfo_vrootx ()
Return the x offset of the virtual root relative to the root window of the screen of this widget.

winfo_vrooty ()
Return the y offset of the virtual root relative to the root window of the screen of this widget.

winfo_width ()
Return the width of this widget.

winfo_x ()
Return the x coordinate of the upper left corner of this widget in the parent.

winfo_y ()
Return the y coordinate of the upper left corner of this widget in the parent.

withdraw ()
Withdraw this widget from the screen such that it is unmapped and forgotten by the window manager. Re-draw it with wm_deiconify.

wm_aspect (minNumer=None, minDenom=None, maxNumer=None, maxDenom=None)
Instruct the window manager to set the aspect ratio (width/height) of this widget to be between MINNUMER/MINDENOM and MAXNUMER/MAXDENOM. Return a tuple of the actual values if no argument is given.

wm_attributes (*args)
This subcommand returns or sets platform specific attributes
The first form returns a list of the platform specific flags and their values. The second form returns the value for the specific option. The third form sets one or more of the values. The values are as follows:
On Windows, -disabled gets or sets whether the window is in a disabled state. -toolwindow gets or sets the style of the window to toolwindow (as defined in the MSDN). -topmost gets or sets whether this is a topmost window (displays above all other windows).
On Macintosh, XXXXX
On Unix, there are currently no special attribute values.

wm_client (name=None)
Store NAME in WM_CLIENT_MACHINE property of this widget. Return current value.

wm_colormapwindows (*wlist)
Store list of window names (WLIST) into WM_COLORMAPWINDOWS property of this widget. This list contains windows whose colormaps differ from their parents. Return current list of widgets if WLIST is empty.

wm_command (value=None)
Store VALUE in WM_COMMAND property. It is the command which shall be used to invoke the application. Return current command if VALUE is None.

wm_deiconify ()
Deiconify this widget. If it was never mapped it will not be mapped. On Windows it will raise this widget and give it the focus.
wm_focusmodel (model=None)
Set focus model to MODEL. “active” means that this widget will claim the focus itself, “passive” means that the window manager shall give the focus. Return current focus model if MODEL is None.

wm_frame ()
Return identifier for decorative frame of this widget if present.

wm_geometry (newGeometry=None)
Set geometry to NEWGEOMETRY of the form =widthxheight+x+y. Return current value if None is given.

wm_grid (baseWidth=None, baseHeight=None, widthInc=None, heightInc=None)
Instruct the window manager that this widget shall only be resized on grid boundaries. WIDTHINC and HEIGHTINC are the width and height of a grid unit in pixels. BASEWIDTH and BASEHEIGHT are the number of grid units requested in Tk_GeometryRequest.

wm_group (pathName=None)
Set the group leader widgets for related widgets to PATHNAME. Return the group leader of this widget if None is given.

wm_iconbitmap (bitmap=None, default=None)
Set bitmap for the iconified widget to BITMAP. Return the bitmap if None is given.
Under Windows, the DEFAULT parameter can be used to set the icon for the widget and any descendants that don’t have an icon set explicitly. DEFAULT can be the relative path to a .ico file (example: root.iconbitmap(default=’myicon.ico’)). See Tk documentation for more information.

wm_iconify ()
Display widget as icon.

wm_iconmask (bitmap=None)
Set mask for the icon bitmap of this widget. Return the mask if None is given.

wm_iconname (newName=None)
Set the name of the icon for this widget. Return the name if None is given.

wm_iconposition (x=None, y=None)
Set the position of the icon of this widget to X and Y. Return a tuple of the current values of X and X if None is given.

wm_iconwindow (pathName=None)
Set widget PATHNAME to be displayed instead of icon. Return the current value if None is given.

wm_maxsize (width=None, height=None)
Set max WIDTH and HEIGHT for this widget. If the window is gridded the values are given in grid units. Return the current values if None is given.

wm_minsize (width=None, height=None)
Set min WIDTH and HEIGHT for this widget. If the window is gridded the values are given in grid units. Return the current values if None is given.

wm_overrideredirect (boolean=None)
Instruct the window manager to ignore this widget if BOOLEAN is given with 1. Return the current value if None is given.

wm_positionfrom (who=None)
Instruct the window manager that the position of this widget shall be defined by the user if WHO is “user”, and by its own policy if WHO is “program”.

wm_protocol (name=None, func=None)
Bind function FUNC to command NAME for this widget. Return the function bound to NAME if None is given. NAME could be e.g. “WM_SAVE_YOURSELF” or “WM_DELETE_WINDOW”.

4.7. libraries Package 345
**wm_resizable**(width=None, height=None)
Instruct the window manager whether this width can be resized in width or height. Both values are boolean values.

**wm_sizefrom**(who=None)
Instruct the window manager that the size of this widget shall be defined by the user if WHO is “user”, and by its own policy if WHO is “program”.

**wm_state**(newstate=None)
Query or set the state of this widget as one of normal, icon, iconic (see wm_iconwindow), withdrawn, or zoomed (Windows only).

**wm_title**(string=None)
Set the title of this widget.

**wm_transient**(master=None)
Instruct the window manager that this widget is transient with regard to widget master.

**wm_withdraw**()
Withdraw this widget from the screen such that it is unmapped and forgotten by the window manager. Re-draw it with wm_deiconify.

**class** robot.libraries.dialogs_py.SelectionDialog(message, values)
Bases: robot.libraries.dialogs_py._TkDialog

**after**(ms, func=None, *args)
Call function once after given time.

MS specifies the time in milliseconds. FUNC gives the function which shall be called. Additional parameters are given as parameters to the function call. Return identifier to cancel scheduling with after_cancel.

**after_cancel**(id)
Cancel scheduling of function identified with ID.

Identifier returned by after or after_idle must be given as first parameter.

**after_idle**(func, *args)
Call FUNC once if the Tcl main loop has no event to process.

Return an identifier to cancel the scheduling with after_cancel.

**aspect**(minNumer=None, minDenom=None, maxNumer=None, maxDenom=None)
Instruct the window manager to set the aspect ratio (width/height) of this widget to be between MINNU-MER/MINDENOM and MAXNUMER/MAXDENOM. Return a tuple of the actual values if no argument is given.

**attributes**(args)
This subcommand returns or sets platform specific attributes

The first form returns a list of the platform specific flags and their values. The second form returns the value for the specific option. The third form sets one or more of the values. The values are as follows:

On Windows, -disabled gets or sets whether the window is in a disabled state. -toolwindow gets or sets the style of the window to toolwindow (as defined in the MSDN). -topmost gets or sets whether this is a topmost window (displays above all other windows).

On Macintosh, XXXXXX

On Unix, there are currently no special attribute values.

**bbox**(column=None, row=None, col2=None, row2=None)
Return a tuple of integer coordinates for the bounding box of this widget controlled by the geometry manager grid.
If COLUMN, ROW is given the bounding box applies from the cell with row and column 0 to the specified cell. If COL2 and ROW2 are given the bounding box starts at that cell.

The returned integers specify the offset of the upper left corner in the master widget and the width and height.

bell (*displayof=0*)
Ring a display’s bell.

bind (*sequence=None, func=None, add=None*)
Bind to this widget at event SEQUENCE a call to function FUNC.

SEQUENCE is a string of concatenated event patterns. An event pattern is of the form <MODIFIER-MODIFIER-TYPE-DETAIL> where MODIFIER is one of Control, Mod2, M2, Shift, Mod3, M3, Lock, Mod4, M4, Button1, B1, Mod5, M5 Button2, B2, Meta, M, Button3, B3, Alt, Button4, B4, Double, Button5, B5 Triple, Mod1, M1. TYPE is one of Activate, Enter, Map, ButtonPress, Button, Exposure, Motion, ButtonRelease FocusIn, MouseWheel, Circulate, FocusOut, Property, Colormap, Gravity Reparent, Configure, KeyPress, Key, Unmap, Deactivate, KeyRelease Visibility, Destroy, Leave and DETAIL is the button number for ButtonPress, ButtonRelease and DETAIL is the Keysym for KeyPress and KeyRelease. Examples are <Control-Button-1> for pressing Control and mouse button 1 or <Alt-A> for pressing A and the Alt key (KeyPress can be omitted). An event pattern can also be a virtual event of the form <<AString>> where AString can be arbitrary. This event can be generated by event_generate. If events are concatenated they must appear shortly after each other.

FUNC will be called if the event sequence occurs with an instance of Event as argument. If the return value of FUNC is “break” no further bound function is invoked.

An additional boolean parameter ADD specifies whether FUNC will be called additionally to the other bound function or whether it will replace the previous function.

Bind will return an identifier to allow deletion of the bound function with unbind without memory leak.

If FUNC or SEQUENCE is omitted the bound function or list of bound events are returned.

bind_all (*sequence=None, func=None, add=None*)
Bind to all widgets at an event SEQUENCE a call to function FUNC. An additional boolean parameter ADD specifies whether FUNC will be called additionally to the other bound function or whether it will replace the previous function. See bind for the return value.

bind_class (*className, sequence=None, func=None, add=None*)
Bind to widgets with bindtag CLASSNAME at event SEQUENCE a call of function FUNC. An additional boolean parameter ADD specifies whether FUNC will be called additionally to the other bound function or whether it will replace the previous function. See bind for the return value.

bindtags (*tagList=None*)
Set or get the list of bindtags for this widget.

With no argument return the list of all bindtags associated with this widget. With a list of strings as argument the bindtags are set to this list. The bindtags determine in which order events are processed (see bind).

cget (*key*)
Return the resource value for a KEY given as string.

client (*name=None*)
Store NAME in WM_CLIENT_MACHINE property of this widget. Return current value.

clipboard_append (*string, **kw*)
Append STRING to the Tk clipboard.

A widget specified at the optional displayof keyword argument specifies the target display. The clipboard can be retrieved with selection_get.
**clipboard_clear(** **kw **)**
Clear the data in the Tk clipboard.
A widget specified for the optional display of keyword argument specifies the target display.

**clipboard_get(** **kw **)**
Retrieve data from the clipboard on window’s display.
The window keyword defaults to the root window of the Tkinter application.
The type keyword specifies the form in which the data is to be returned and should be an atom name such as STRING or FILE_NAME. Type defaults to STRING, except on X11, where the default is to try UTF8_STRING and fall back to STRING.
This command is equivalent to:

```
selection_get(CLIPBOARD)
```

**colormapwindows (** *wlist **)**
Store list of window names (WLIST) into WM_COLORMAPWINDOWS property of this widget. This list contains windows whose colormaps differ from their parents. Return current list of widgets if WLIST is empty.

**colormodel (** value=None **)**
Useless. Not implemented in Tk.

**columnconfigure (** index, cnf={}, **kw **)**
Configure column INDEX of a grid.
Valid resources are minsize (minimum size of the column), weight (how much does additional space propagate to this column) and pad (how much space to let additionally).

**command (** value=None **)**
Store VALUE in WM_COMMAND property. It is the command which shall be used to invoke the application. Return current command if VALUE is None.

**config (** cnf=None, **kw **)**
Configure resources of a widget.
The values for resources are specified as keyword arguments. To get an overview about the allowed keyword arguments call the method keys.

**configure (** cnf=None, **kw **)**
Configure resources of a widget.
The values for resources are specified as keyword arguments. To get an overview about the allowed keyword arguments call the method keys.

**deiconify ()**
Deiconify this widget. If it was never mapped it will not be mapped. On Windows it will raise this widget and give it the focus.

**deletecommand (** name **)**
Internal function.
Delete the Tcl command provided in NAME.

**destroy ()**
Destroy this and all descendants widgets.

**event_add (** virtual, *sequences **)**
Bind a virtual event VIRTUAL (of the form <<Name>>) to an event SEQUENCE such that the virtual event is triggered whenever SEQUENCE occurs.
**event_delete**(virtual, *sequences)

Unbind a virtual event VIRTUAL from SEQUENCE.

**event_generate**(sequence, **kw)

Generate an event SEQUENCE. Additional keyword arguments specify parameter of the event (e.g. x, y, rootx, rooty).

**event_info**(virtual=\None)

Return a list of all virtual events or the information about the SEQUENCE bound to the virtual event VIRTUAL.

**focus**()

Direct input focus to this widget.

If the application currently does not have the focus this widget will get the focus if the application gets the focus through the window manager.

**focus_displayof**()

Return the widget which has currently the focus on the display where this widget is located.

Return None if the application does not have the focus.

**focus_force**()

Direct input focus to this widget even if the application does not have the focus. Use with caution!

**focus_get**()

Return the widget which has currently the focus in the application.

Use focus_displayof to allow working with several displays. Return None if application does not have the focus.

**focus_lastfor**()

Return the widget which would have the focus if top level for this widget gets the focus from the window manager.

**focus_set**()

Direct input focus to this widget.

If the application currently does not have the focus this widget will get the focus if the application gets the focus through the window manager.

**focusmodel**(model=\None)

Set focus model to MODEL. “active” means that this widget will claim the focus itself, “passive” means that the window manager shall give the focus. Return current focus model if MODEL is None.

**frame**()

Return identifier for decorative frame of this widget if present.

**geometry**(newGeometry=\None)

Set geometry to NEWGEOMETRY of the form =widthxheight+x+y. Return current value if None is given.

**getboolean**(s)

Return a boolean value for Tcl boolean values true and false given as parameter.

**getdouble**

alias of float

**getint**

alias of int

**getvar**(name='PY_VAR')

Return value of Tcl variable NAME.
grab_current()
Return widget which has currently the grab in this application or None.

grab_release()
Release grab for this widget if currently set.

grab_set()
Set grab for this widget.
A grab directs all events to this and descendant widgets in the application.

grab_set_global()
Set global grab for this widget.
A global grab directs all events to this and descendant widgets on the display. Use with caution - other
applications do not get events anymore.

grab_status()
Return None, “local” or “global” if this widget has no, a local or a global grab.

grid(baseWidth=None, baseHeight=None, widthInc=None, heightInc=None)
Instruct the window manager that this widget shall only be resized on grid boundaries. WIDTHINC and
HEIGHTINC are the width and height of a grid unit in pixels. BASEWIDTH and BASEHEIGHT are the
number of grid units requested in Tk_GeometryRequest.

grid_bbox(column=None, row=None, col2=None, row2=None)
Return a tuple of integer coordinates for the bounding box of this widget controlled by the geometry
manager grid.

If COLUMN, ROW is given the bounding box applies from the cell with row and column 0 to the specified
cell. If COL2 and ROW2 are given the bounding box starts at that cell.

The returned integers specify the offset of the upper left corner in the master widget and the width and
height.

grid_columnconfigure(index, cnf={}, **kw)
Configure column INDEX of a grid.

Valid resources are minsize (minimum size of the column), weight (how much does additional space
propagate to this column) and pad (how much space to let additionally).

grid_location(x, y)
Return a tuple of column and row which identify the cell at which the pixel at position X and Y inside the
master widget is located.

grid_propagate(flag=['_noarg_'])
Set or get the status for propagation of geometry information.

A boolean argument specifies whether the geometry information of the slaves will determine the size of
this widget. If no argument is given, the current setting will be returned.

grid_rowconfigure(index, cnf={}, **kw)
Configure row INDEX of a grid.

Valid resources are minsize (minimum size of the row), weight (how much does additional space propagate
to this row) and pad (how much space to let additionally).

grid_size()
Return a tuple of the number of column and rows in the grid.

grid_slaves(row=None, column=None)
Return a list of all slaves of this widget in its packing order.
group (pathName=None)
Set the group leader widgets for related widgets to PATHNAME. Return the group leader of this widget if None is given.

iconbitmap (bitmap=None, default=None)
Set bitmap for the iconified widget to BITMAP. Return the bitmap if None is given.

Under Windows, the DEFAULT parameter can be used to set the icon for the widget and any descendants that don’t have an icon set explicitly. DEFAULT can be the relative path to a .ico file (example: root.iconbitmap(default=’myicon.ico’)). See Tk documentation for more information.

iconify ()
Display widget as icon.

iconmask (bitmap=None)
Set mask for the icon bitmap of this widget. Return the mask if None is given.

iconname (newName=None)
Set the name of the icon for this widget. Return the name if None is given.

iconposition (x=None, y=None)
Set the position of the icon of this widget to X and Y. Return a tuple of the current values of X and Y if None is given.

iconwindow (pathName=None)
Set widget PATHNAME to be displayed instead of icon. Return the current value if None is given.

image_names ()
Return a list of all existing image names.

image_types ()
Return a list of all available image types (e.g. photo bitmap).

keys ()
Return a list of all resource names of this widget.

lift (aboveThis=None)
Raise this widget in the stacking order.

lower (belowThis=None)
Lower this widget in the stacking order.

mainloop (n=0)
Call the mainloop of Tk.

maxsize (width=None, height=None)
Set max WIDTH and HEIGHT for this widget. If the window is gridded the values are given in grid units. Return the current values if None is given.

minsize (width=None, height=None)
Set min WIDTH and HEIGHT for this widget. If the window is gridded the values are given in grid units. Return the current values if None is given.

nametowidget (name)
Return the Tkinter instance of a widget identified by its Tcl name NAME.

option_add (pattern, value, priority=None)
Set a VALUE (second parameter) for an option PATTERN (first parameter).

An optional third parameter gives the numeric priority (defaults to 80).

option_clear ()
Clear the option database.
It will be reloaded if option_add is called.

**option_get**(name, className)
Return the value for an option NAME for this widget with CLASSNAME.
Values with higher priority override lower values.

**option_readfile**(fileName, priority=None)
Read file FILENAME into the option database.
An optional second parameter gives the numeric priority.

**overrideredirect**(boolean=None)
Instruct the window manager to ignore this widget if BOOLEAN is given with 1. Return the current value if None is given.

**pack_propagate**(flag=[_noarg_])
Set or get the status for propagation of geometry information.
A boolean argument specifies whether the geometry information of the slaves will determine the size of this widget. If no argument is given the current setting will be returned.

**pack_slaves**()
Return a list of all slaves of this widget in its packing order.

**place_slaves**()
Return a list of all slaves of this widget in its packing order.

**positionfrom**(who=None)
Instruct the window manager that the position of this widget shall be defined by the user if WHO is “user”, and by its own policy if WHO is “program”.

**propagate**(flag=[_noarg_])
Set or get the status for propagation of geometry information.
A boolean argument specifies whether the geometry information of the slaves will determine the size of this widget. If no argument is given the current setting will be returned.

**protocol**(name=None, func=None)
Bind function FUNC to command NAME for this widget. Return the function bound to NAME if None is given. NAME could be e.g. “WM_SAVE_YOURSELF” or “WM_DELETE_WINDOW”.

**quit**( )
Quit the Tcl interpreter. All widgets will be destroyed.

**register**(func, subst=None, needcleanup=1)
Return a newly created Tcl function. If this function is called, the Python function FUNC will be executed. An optional function SUBST can be given which will be executed before FUNC.

**resizable**(width=None, height=None)
Instruct the window manager whether this width can be resized in WIDTH or HEIGHT. Both values are boolean values.

**rowconfigure**(index, cnf={}, **kw)
Configure row INDEX of a grid.
Valid resources are minsize (minimum size of the row), weight (how much does additional space propagate to this row) and pad (how much space to let additionally).

**selection_clear**( **kw)
Clear the current X selection.

**selection_get**( **kw)
Return the contents of the current X selection.
A keyword parameter selection specifies the name of the selection and defaults to PRIMARY. A keyword parameter displayof specifies a widget on the display to use. A keyword parameter type specifies the form of data to be fetched, defaulting to STRING except on X11, where UTF8_STRING is tried before STRING.

**selection_handle**(command, **kw)
Specify a function COMMAND to call if the X selection owned by this widget is queried by another application.

This function must return the contents of the selection. The function will be called with the arguments OFFSET and LENGTH which allows the chunking of very long selections. The following keyword parameters can be provided: selection - name of the selection (default PRIMARY), type - type of the selection (e.g. STRING, FILE_NAME).

**selection_own**(**kw)
Become owner of X selection.

A keyword parameter selection specifies the name of the selection (default PRIMARY).

**selection_own_get**(**kw)
Return owner of X selection.

The following keyword parameter can be provided: selection - name of the selection (default PRIMARY), type - type of the selection (e.g. STRING, FILE_NAME).

**send**(interp, cmd, *args)
Send Tcl command CMD to different interpreter INTERP to be executed.

**setvar**(name='PY_VAR', value='1')
Set Tcl variable NAME to VALUE.

**show**( )

**size**( )
Return a tuple of the number of column and rows in the grid.

**sizefrom**(who=None)
Instruct the window manager that the size of this widget shall be defined by the user if WHO is “user”, and by its own policy if WHO is “program”.

**slaves**( )
Return a list of all slaves of this widget in its packing order.

**state**(newstate=None)
Query or set the state of this widget as one of normal, icon, iconic (see wm_iconwindow), withdrawn, or zoomed (Windows only).

**title**(string=None)
Set the title of this widget.

**tk_bisque**( )
Change the color scheme to light brown as used in Tk 3.6 and before.

**tk_focusFollowsMouse**( )
The widget under mouse will get automatically focus. Can not be disabled easily.

**tk_focusNext**( )
Return the next widget in the focus order which follows widget which has currently the focus.

The focus order first goes to the next child, then to the children of the child recursively and then to the next sibling which is higher in the stacking order. A widget is omitted if it has the takefocus resource set to 0.
Robot Framework Documentation, Release 2.7.7

```python

tk_focusPrev()
   Return previous widget in the focus order. See tk_focusNext for details.

tk_menuBar(*args)
   Do not use. Needed in Tk 3.6 and earlier.

tk_setPalette(*args, **kw)
   Set a new color scheme for all widget elements.
   A single color as argument will cause that all colors of Tk widget elements are derived from this. Alternatively several keyword parameters and its associated colors can be given. The following keywords are valid: activeBackground, foreground, selectColor, activeForeground, highlightBackground, selectBackground, background, highlightColor, selectForeground, disabledForeground, insertBackground, troughColor.

tk_strictMotif(boolean=None)
   Set Tcl internal variable, whether the look and feel should adhere to Motif.
   A parameter of 1 means adhere to Motif (e.g. no color change if mouse passes over slider). Returns the set value.

tkraise(aboveThis=None)
   Raise this widget in the stacking order.

transient(master=None)
   Instruct the window manager that this widget is transient with regard to widget MASTER.

unbind(sequence, funcid=None)
   Unbind for this widget for event SEQUENCE the function identified with FUNCID.

unbind_all(sequence)
   Unbind for all widgets for event SEQUENCE all functions.

unbind_class(className, sequence)
   Unbind for all widgets with bindtag CLASSNAME for event SEQUENCE all functions.

update()
   Enter event loop until all pending events have been processed by Tcl.

update_idletasks()
   Enter event loop until all idle callbacks have been called. This will update the display of windows but not process events caused by the user.

wait_variable(name='PY_VAR')
   Wait until the variable is modified.
   A parameter of type IntVar, StringVar, DoubleVar or BooleanVar must be given.

wait_visibility(window=None)
   Wait until the visibility of a WIDGET changes (e.g. it appears).
   If no parameter is given self is used.

wait_window(window=None)
   Wait until a WIDGET is destroyed.
   If no parameter is given self is used.

waitvar(name='PY_VAR')
   Wait until the variable is modified.
   A parameter of type IntVar, StringVar, DoubleVar or BooleanVar must be given.
```
wininfo_atom(name, displayof=0)
    Return integer which represents atom NAME.

wininfo_atomname(id, displayof=0)
    Return name of atom with identifier ID.

wininfo_cells()
    Return number of cells in the colormap for this widget.

wininfo_children()
    Return a list of all widgets which are children of this widget.

wininfo_class()
    Return window class name of this widget.

wininfo_colormapfull()
    Return true if at the last color request the colormap was full.

wininfo_containing(rootX, rootY, displayof=0)
    Return the widget which is at the root coordinates ROOTX, ROOTY.

wininfo_depth()
    Return the number of bits per pixel.

wininfo_exists()
    Return true if this widget exists.

wininfo_fpixels(number)
    Return the number of pixels for the given distance NUMBER (e.g. “3c”) as float.

wininfo_geometry()
    Return geometry string for this widget in the form “widthxheight+X+Y”.

wininfo_height()
    Return height of this widget.

wininfo_id()
    Return identifier ID for this widget.

wininfo_interps(displayof=0)
    Return the name of all Tcl interpreters for this display.

wininfo_ismapped()
    Return true if this widget is mapped.

wininfo_manager()
    Return the window mananger name for this widget.

wininfo_name()
    Return the name of this widget.

wininfo_parent()
    Return the name of the parent of this widget.

wininfo_pathname(id, displayof=0)
    Return the pathname of the widget given by ID.

wininfo_pixels(number)
    Rounded integer value of wininfo_fpixels.

wininfo_pointerx()
    Return the x coordinate of the pointer on the root window.
Robot Framework Documentation, Release 2.7.7

```

winfo_pointerxy()
Return a tuple of x and y coordinates of the pointer on the root window.

winfo_pointery()
Return the y coordinate of the pointer on the root window.

winfo_reqheight()
Return requested height of this widget.

winfo_reqwidth()
Return requested width of this widget.

winfo_rgb(color)
Return tuple of decimal values for red, green, blue for COLOR in this widget.

winfo_rootx()
Return x coordinate of upper left corner of this widget on the root window.

winfo_rooty()
Return y coordinate of upper left corner of this widget on the root window.

winfo_screen()
Return the screen name of this widget.

winfo_screencells()
Return the number of the cells in the colormap of the screen of this widget.

winfo_screendepth()
Return the number of bits per pixel of the root window of the screen of this widget.

winfo_screenheight()
Return the number of pixels of the height of the screen of this widget in pixel.

winfo_screenmmheight()
Return the number of pixels of the height of the screen of this widget in mm.

winfo_screenmmwidth()
Return the number of pixels of the width of the screen of this widget in mm.

winfo_screenvisual()
Return one of the strings directcolor, grayscale, pseudocolor, staticcolor, staticgray, or truecolor for the
default colormodel of this screen.

winfo_screenwidth()
Return the number of pixels of the width of the screen of this widget in pixel.

winfo_server()
Return information of the X-Server of the screen of this widget in the form “XmajorRminor vendor vendorVersion”.

winfo_toplevel()
Return the toplevel widget of this widget.

winfo_viewable()
Return true if the widget and all its higher ancestors are mapped.

winfo_visual()
Return one of the strings directcolor, grayscale, pseudocolor, staticcolor, staticgray, or truecolor for the
colormodel of this widget.

winfo_visualid()
Return the X identifier for the visual for this widget.
```
Robot Framework Documentation, Release 2.7.7

winfo_visualsavailable (includeids=0)
Return a list of all visuals available for the screen of this widget.

Each item in the list consists of a visual name (see winfo_visual), a depth and if INCLUDEIDS=1 is given also the X identifier.

winfo_vrootheight()
Return the height of the virtual root window associated with this widget in pixels. If there is no virtual root window return the height of the screen.

winfo_vrootwidth()
Return the width of the virtual root window associated with this widget in pixel. If there is no virtual root window return the width of the screen.

winfo_vrootx()
Return the x offset of the virtual root relative to the root window of the screen of this widget.

winfo_vrooty()
Return the y offset of the virtual root relative to the root window of the screen of this widget.

winfo_width()
Return the width of this widget.

winfo_x()
Return the x coordinate of the upper left corner of this widget in the parent.

winfo_y()
Return the y coordinate of the upper left corner of this widget in the parent.

withdraw()
Withdraw this widget from the screen such that it is unmapped and forgotten by the window manager. Re-draw it with wm_deiconify.

wm_aspect (minNumer=None, minDenom=None, maxNumer=None, maxDenom=None)
Instruct the window manager to set the aspect ratio (width/height) of this widget to be between MINNUMER/MINDENOM and MAXNUMER/MAXDENOM. Return a tuple of the actual values if no argument is given.

wm_attributes (*args)
This subcommand returns or sets platform specific attributes

The first form returns a list of the platform specific flags and their values. The second form returns the value for the specific option. The third form sets one or more of the values. The values are as follows:

On Windows, -disabled gets or sets whether the window is in a disabled state. -toolwindow gets or sets the style of the window to toolwindow (as defined in the MSDN). -topmost gets or sets whether this is a topmost window (displays above all other windows).

On Macintosh, XXXXX

On Unix, there are currently no special attribute values.

wm_client (name=None)
Store NAME in WM_CLIENT_MACHINE property of this widget. Return current value.

wm_colormapwindows (*wlist)
Store list of window names (WLIST) into WM_COLORMAPWINDOWS property of this widget. This list contains windows whose colormaps differ from their parents. Return current list of widgets if WLIST is empty.

wm_command (value=None)
Store VALUE in WM_COMMAND property. It is the command which shall be used to invoke the application. Return current command if VALUE is None.
wm_deiconify()
Deiconify this widget. If it was never mapped it will not be mapped. On Windows it will raise this widget
and give it the focus.

wm_focusmodel (model=None)
Set focus model to MODEL. “active” means that this widget will claim the focus itself, “passive” means
that the window manager shall give the focus. Return current focus model if MODEL is None.

wm_frame()
Return identifier for decorative frame of this widget if present.

wm_geometry (newGeometry=None)
Set geometry to NEWGEOMETRY of the form =widthxheight+x+y. Return current value if None is given.

wm_grid (baseWidth=None, baseHeight=None, widthInc=None, heightInc=None)
Instruct the window manager that this widget shall only be resized on grid boundaries. WIDTHINC and
HEIGHTINC are the width and height of a grid unit in pixels. BASEWIDTH and BASEHEIGHT are the
number of grid units requested in Tk_GeometryRequest.

wm_group (pathName=None)
Set the group leader widgets for related widgets to PATHNAME. Return the group leader of this widget if
None is given.

wm_iconbitmap (bitmap=None, default=None)
Set bitmap for the iconified widget to BITMAP. Return the bitmap if None is given.
Under Windows, the DEFAULT parameter can be used to set the icon for the widget and any descen-
dents that don’t have an icon set explicitly. DEFAULT can be the relative path to a .ico file (example:
root.iconbitmap(default=’myicon.ico’) ). See Tk documentation for more information.

wm_iconify()
Display widget as icon.

wm_iconmask (bitmap=None)
Set mask for the icon bitmap of this widget. Return the mask if None is given.

wm_iconname (newName=None)
Set the name of the icon for this widget. Return the name if None is given.

wm_iconposition (x=None, y=None)
Set the position of the icon of this widget to X and Y. Return a tuple of the current values of X and X if
None is given.

wm_iconwindow (pathName=None)
Set widget PATHNAME to be displayed instead of icon. Return the current value if None is given.

wm_maxsize (width=None, height=None)
Set max WIDTH and HEIGHT for this widget. If the window is gridded the values are given in grid units.
Return the current values if None is given.

wm_minsize (width=None, height=None)
Set min WIDTH and HEIGHT for this widget. If the window is gridded the values are given in grid units.
Return the current values if None is given.

wm_overrideredirect (boolean=None)
Instruct the window manager to ignore this widget if BOOLEAN is given with 1. Return the current value
if None is given.

wm_positionfrom (who=None)
Instruct the window manager that the position of this widget shall be defined by the user if WHO is “user”,
and by its own policy if WHO is “program”.

358 Chapter 4. All packages
Robot Framework Documentation, Release 2.7.7

wm_protocol (name=None, func=None)
   Bind function FUNC to command NAME for this widget. Return the function bound to NAME if None is given. NAME could be e.g. “WM_SAVE_YOURSELF” or “WM_DELETE_WINDOW”.

wm_resizable (width=None, height=None)
   Instruct the window manager whether this width can be resized in WIDTH or HEIGHT. Both values are boolean values.

wm_sizefrom (who=None)
   Instruct the window manager that the size of this widget shall be defined by the user if WHO is “user”, and by its own policy if WHO is “program”.

wm_state (newstate=None)
   Query or set the state of this widget as one of normal, icon, iconic (see wm_iconwindow), withdrawn, or zoomed (Windows only).

wm_title (string=None)
   Set the title of this widget.

wm_transient (master=None)
   Instruct the window manager that this widget is transient with regard to widget MASTER.

wm_withdraw ()
   Withdraw this widget from the screen such that it is unmapped and forgotten by the window manager. Re-draw it with wm_deiconify.

class robot.libraries.dialogs_py.PassFailDialog (message, value=None)
   Bases: robot.libraries.dialogs_py._TkDialog

after (ms, func=None, *args)
   Call function once after given time.

   MS specifies the time in milliseconds. FUNC gives the function which shall be called. Additional parameters are given as parameters to the function call. Return identifier to cancel scheduling with after_cancel.

after_cancel (id)
   Cancel scheduling of function identified with ID.

   Identifier returned by after or after_idle must be given as first parameter.

after_idle (func, *args)
   Call FUNC once if the Tcl main loop has no event to process.

   Return an identifier to cancel the scheduling with after_cancel.

aspect (minNumer=None, minDenom=None, maxNumer=None, maxDenom=None)
   Instruct the window manager to set the aspect ratio (width/height) of this widget to be between MINNUMER/MINDENOM and MAXNUMER/MAXDENOM. Return a tuple of the actual values if no argument is given.

attributes (*args)
   This subcommand returns or sets platform specific attributes

   The first form returns a list of the platform specific flags and their values. The second form returns the value for the specific option. The third form sets one or more of the values. The values are as follows:

   On Windows, -disabled gets or sets whether the window is in a disabled state. -toolwindow gets or sets the style of the window to toolwindow (as defined in the MSDN). -topmost gets or sets whether this is a topmost window (displays above all other windows).

   On Macintosh, XXXXX

   On Unix, there are currently no special attribute values.

4.7. libraries Package 359
bbox (column=None, row=None, col2=None, row2=None)

Return a tuple of integer coordinates for the bounding box of this widget controlled by the geometry manager grid.

If COLUMN, ROW is given the bounding box applies from the cell with row and column 0 to the specified cell. If COL2 and ROW2 are given the bounding box starts at that cell.

The returned integers specify the offset of the upper left corner in the master widget and the width and height.

bell (displayof=0)

Ring a display’s bell.

bind (sequence=None, func=None, add=None)

Bind to this widget at event SEQUENCE a call to function FUNC.

SEQUENCE is a string of concatenated event patterns. An event pattern is of the form <MODIFIER-MODIFIER-TYPE-DETAIL> where MODIFIER is one of Control, Mod2, M2, Shift, Mod3, M3, Lock, Mod4, M4, Button1, B1, Mod5, M5 Button2, B2, Meta, M, Button3, B3, Alt, Button4, B4, Double, Button5, B5 Triple, Mod1, M1. TYPE is one of Activate, Enter, Map, ButtonPress, Button, Expose, Motion, ButtonRelease FocusIn, MouseWheel, Circulate, FocusOut, Property, Colormap, Gravity Reparent, Configure, KeyPress, Key, Unmap, Deactivate, KeyRelease Visibility, Destroy, Leave and DETAIL is the button number for ButtonPress, ButtonRelease and DETAIL is the Keysym for KeyPress and KeyRelease. Examples are <Control-Button-1> for pressing Control and mouse button 1 or <Alt-A> for pressing A and the Alt key (KeyPress can be omitted). An event pattern can also be a virtual event of the form <<AString>> where AString can be arbitrary. This event can be generated by event_generate. If events are concatenated they must appear shortly after each other.

FUNC will be called if the event sequence occurs with an instance of Event as argument. If the return value of FUNC is “break” no further bound function is invoked.

An additional boolean parameter ADD specifies whether FUNC will be called additionally to the other bound function or whether it will replace the previous function.

Bind will return an identifier to allow deletion of the bound function with unbind without memory leak.

If FUNC or SEQUENCE is omitted the bound function or list of bound events are returned.

bind_all (sequence=None, func=None, add=None)

Bind to all widgets at an event SEQUENCE a call to function FUNC. An additional boolean parameter ADD specifies whether FUNC will be called additionally to the other bound function or whether it will replace the previous function. See bind for the return value.

bind_class (className, sequence=None, func=None, add=None)

Bind to widgets with bindtag CLASSNAME at event SEQUENCE a call of function FUNC. An additional boolean parameter ADD specifies whether FUNC will be called additionally to the other bound function or whether it will replace the previous function. See bind for the return value.

bindtags (tagList=None)

Set or get the list of bindtags for this widget.

With no argument return the list of all bindtags associated with this widget. With a list of strings as argument the bindtags are set to this list. The bindtags determine in which order events are processed (see bind).

cget (key)

Return the resource value for a KEY given as string.

client (name=None)

Store NAME in WM_CLIENT_MACHINE property of this widget. Return current value.
clipboard_append(string, **kw)
Append STRING to the Tk clipboard.
A widget specified at the optional displayof keyword argument specifies the target display. The clipboard
can be retrieved with selection_get.

clipboard_clear(**kw)
Clear the data in the Tk clipboard.
A widget specified for the optional displayof keyword argument specifies the target display.

clipboard_get(**kw)
Retrieve data from the clipboard on window’s display.
The window keyword defaults to the root window of the Tkinter application.
The type keyword specifies the form in which the data is to be returned and should be an atom name
such as STRING or FILE_NAME. Type defaults to STRING, except on X11, where the default is to try
UTF8_STRING and fall back to STRING.
This command is equivalent to:
selection_get(CLIPBOARD)

colormapwindows(*wlist)
Store list of window names (WLIST) into WM_COLORMAPWINDOWS property of this widget. This
list contains windows whose colormaps differ from their parents. Return current list of widgets if WLIST
is empty.

colormodel(value=None)
Useless. Not implemented in Tk.

columnconfigure(index, cnf={}, **kw)
Configure column INDEX of a grid.
Valid resources are minsize (minimum size of the column), weight (how much does additional space
propagate to this column) and pad (how much space to let additionally).

command(value=None)
Store VALUE in WM_COMMAND property. It is the command which shall be used to invoke the appli-
cation. Return current command if VALUE is None.

config(cnf=None, **kw)
Configure resources of a widget.
The values for resources are specified as keyword arguments. To get an overview about the allowed
keyword arguments call the method keys.

configure(cnf=None, **kw)
Configure resources of a widget.
The values for resources are specified as keyword arguments. To get an overview about the allowed
keyword arguments call the method keys.

deiconify()
Deiconify this widget. If it was never mapped it will not be mapped. On Windows it will raise this widget
and give it the focus.

deletecommand(name)
Internal function.
Delete the Tcl command provided in NAME.
**destroy()**
Destroy this and all descendants widgets.

**event_add**(virtual, *sequences)**
Bind a virtual event VIRTUAL (of the form <<Name>>) to an event SEQUENCE such that the virtual event is triggered whenever SEQUENCE occurs.

**event_delete**(virtual, *sequences)**
Unbind a virtual event VIRTUAL from SEQUENCE.

**event_generate**(sequence, **kw)**
Generate an event SEQUENCE. Additional keyword arguments specify parameter of the event (e.g. x, y, rootx, rooty).

**event_info**(virtual=None)**
Return a list of all virtual events or the information about the SEQUENCE bound to the virtual event VIRTUAL.

**focus()**
Direct input focus to this widget.

If the application currently does not have the focus this widget will get the focus if the application gets the focus through the window manager.

**focus_displayof()**
Return the widget which has currently the focus on the display where this widget is located.

Return None if the application does not have the focus.

**focus_force()**
Direct input focus to this widget even if the application does not have the focus. Use with caution!

**focus_get()**
Return the widget which has currently the focus in the application.

Use focus_displayof to allow working with several displays. Return None if application does not have the focus.

**focus_lastfor()**
Return the widget which would have the focus if top level for this widget gets the focus from the window manager.

**focus_set()**
Direct input focus to this widget.

If the application currently does not have the focus this widget will get the focus if the application gets the focus through the window manager.

**focusmodel**(model=None)**
Set focus model to MODEL. “active” means that this widget will claim the focus itself, “passive” means that the window manager shall give the focus. Return current focus model if MODEL is None.

**frame()**
Return identifier for decorative frame of this widget if present.

**geometry**(newGeometry=None)**
Set geometry to NEWGEOMETRY of the form =widthxheight+x+y. Return current value if None is given.

**getboolean**(s)**
Return a boolean value for Tcl boolean values true and false given as parameter.

**getdouble**
alias of float
getint
   alias of int

getvar (name='PY_VAR')
   Return value of Tcl variable NAME.

grab_current()
   Return widget which has currently the grab in this application or None.

grab_release()
   Release grab for this widget if currently set.

grab_set()
   Set grab for this widget.

   A grab directs all events to this and descendant widgets in the application.

grab_set_global()
   Set global grab for this widget.

   A global grab directs all events to this and descendant widgets on the display. Use with caution - other
   applications do not get events anymore.

grab_status()
   Return None, “local” or “global” if this widget has no, a local or a global grab.

grid (baseWidth=None, baseHeight=None, widthInc=None, heightInc=None)
   Instruct the window manager that this widget shall only be resized on grid boundaries. WIDTHINC and
   HEIGHTINC are the width and height of a grid unit in pixels. BASEWIDTH and BASEHEIGHT are the
   number of grid units requested in Tk_GeometryRequest.

grid_bbox (column=None, row=None, col2=None, row2=None)
   Return a tuple of integer coordinates for the bounding box of this widget controlled by the geometry
   manager grid.

   If COLUMN, ROW is given the bounding box applies from the cell with row and column 0 to the specified
   cell. If COL2 and ROW2 are given the bounding box starts at that cell.

   The returned integers specify the offset of the upper left corner in the master widget and the width and
   height.

grid_columnconfigure (index, cnf={}, **kw)
   Configure column INDEX of a grid.

   Valid resources are minsize (minimum size of the column), weight (how much does additional space
   propagate to this column) and pad (how much space to let additionally).

grid_location (x, y)
   Return a tuple of column and row which identify the cell at which the pixel at position X and Y inside the
   master widget is located.

grid_propagate (flag='__noarg__')
   Set or get the status for propagation of geometry information.

   A boolean argument specifies whether the geometry information of the slaves will determine the size of
   this widget. If no argument is given, the current setting will be returned.

grid_rowconfigure (index, cnf={}, **kw)
   Configure row INDEX of a grid.

   Valid resources are minsize (minimum size of the row), weight (how much does additional space propagate
to this row) and pad (how much space to let additionally).
grid_size()
    Return a tuple of the number of column and rows in the grid.

grid_slaves (row=None, column=None)
    Return a list of all slaves of this widget in its packing order.

group (pathName=None)
    Set the group leader widgets for related widgets to PATHNAME. Return the group leader of this widget if
    None is given.

iconbitmap (bitmap=None, default=None)
    Set bitmap for the iconified widget to BITMAP. Return the bitmap if None is given.
    Under Windows, the DEFAULT parameter can be used to set the icon for the widget and any descen-
    dents that don’t have an icon set explicitly. DEFAULT can be the relative path to a .ico file (example:
    root.iconbitmap(default='myicon.ico' ) ). See Tk documentation for more information.

iconify ()
    Display widget as icon.

iconmask (bitmap=None)
    Set mask for the icon bitmap of this widget. Return the mask if None is given.

iconname (newName=None)
    Set the name of the icon for this widget. Return the name if None is given.

iconposition (x=None, y=None)
    Set the position of the icon of this widget to X and Y. Return a tuple of the current values of X and X if
    None is given.

iconwindow (pathName=None)
    Set widget PATHNAME to be displayed instead of icon. Return the current value if None is given.

image_names ()
    Return a list of all existing image names.

image_types ()
    Return a list of all available image types (e.g. photo bitmap).

keys ()
    Return a list of all resource names of this widget.

lift (aboveThis=None)
    Raise this widget in the stacking order.

lower (belowThis=None)
    Lower this widget in the stacking order.

mainloop (n=0)
    Call the mainloop of Tk.

maxsize (width=None, height=None)
    Set max WIDTH and HEIGHT for this widget. If the window is gridded the values are given in grid units.
    Return the current values if None is given.

minsize (width=None, height=None)
    Set min WIDTH and HEIGHT for this widget. If the window is gridded the values are given in grid units.
    Return the current values if None is given.

nametowidget (name)
    Return the Tkinter instance of a widget identified by its Tcl name NAME.
option_add (pattern, value, priority=None)
Set a VALUE (second parameter) for an option PATTERN (first parameter).
An optional third parameter gives the numeric priority (defaults to 80).

option_clear ()
Clear the option database.
It will be reloaded if option_add is called.

option_get (name, className)
Return the value for an option NAME for this widget with CLASSNAME.
Values with higher priority override lower values.

option_readfile (fileName, priority=None)
Read file FILENAME into the option database.
An optional second parameter gives the numeric priority.

overrideredirect (boolean=None)
Instruct the window manager to ignore this widget if BOOLEAN is given with 1. Return the current value if None is given.

pack_propagate (flag=['_noarg_'])
Set or get the status for propagation of geometry information.
A boolean argument specifies whether the geometry information of the slaves will determine the size of this widget. If no argument is given the current setting will be returned.

pack_slaves ()
Return a list of all slaves of this widget in its packing order.

place_slaves ()
Return a list of all slaves of this widget in its packing order.

positionfrom (who=None)
Instruct the window manager that the position of this widget shall be defined by the user if WHO is "user", and by its own policy if WHO is "program".

propagate (flag=['_noarg_'])
Set or get the status for propagation of geometry information.
A boolean argument specifies whether the geometry information of the slaves will determine the size of this widget. If no argument is given the current setting will be returned.

protocol (name=None, func=None)
Bind function FUNC to command NAME for this widget. Return the function bound to NAME if None is given. NAME could be e.g. "WM_SAVE_YOURSELF" or "WM_DELETE_WINDOW".

quit ()
Quit the Tcl interpreter. All widgets will be destroyed.

register (func, subst=None, needclean=1)
Return a newly created Tcl function. If this function is called, the Python function FUNC will be executed. An optional function SUBST can be given which will be executed before FUNC.

resizable (width=None, height=None)
Instruct the window manager whether this width can be resized in WIDTH or HEIGHT. Both values are boolean values.

rowconfigure (index, cnf={}, **kw)
Configure row INDEX of a grid.
Valid resources are minsize (minimum size of the row), weight (how much does additional space propagate to this row) and pad (how much space to let additionally).

**selection_clear(**kw**)
Clear the current X selection.

**selection_get(**kw**)
Return the contents of the current X selection.

A keyword parameter selection specifies the name of the selection and defaults to PRIMARY. A keyword parameter displayof specifies a widget on the display to use. A keyword parameter type specifies the form of data to be fetched, defaulting to STRING except on X11, where UTF8_STRING is tried before STRING.

**selection_handle**(command, **kw**)
Specify a function COMMAND to call if the X selection owned by this widget is queried by another application.

This function must return the contents of the selection. The function will be called with the arguments OFFSET and LENGTH which allows the chunking of very long selections. The following keyword parameters can be provided: selection - name of the selection (default PRIMARY), type - type of the selection (e.g. STRING, FILE_NAME).

**selection_own(**kw**)
Become owner of X selection.

A keyword parameter selection specifies the name of the selection (default PRIMARY).

**selection_own_get(**kw**)
Return owner of X selection.

The following keyword parameter can be provided: selection - name of the selection (default PRIMARY), type - type of the selection (e.g. STRING, FILE_NAME).

**send**(interp, cmd, *args)
Send Tcl command CMD to different interpreter INTERP to be executed.

**setvar**(name='PY_VAR', value='1')
Set Tcl variable NAME to VALUE.

**show**()

**size**()
Return a tuple of the number of column and rows in the grid.

**sizefrom**(who=None)
Instruct the window manager that the size of this widget shall be defined by the user if WHO is “user”, and by its own policy if WHO is “program”.

**slaves**()
Return a list of all slaves of this widget in its packing order.

**state**(newstate=None)
Query or set the state of this widget as one of normal, icon, iconic (see wm_iconwindow), withdrawn, or zoomed (Windows only).

**title**(string=None)
Set the title of this widget.

**tk_bisque**()
Change the color scheme to light brown as used in Tk 3.6 and before.
tk_focusFollowsMouse()
The widget under mouse will get automatically focus. Can not be disabled easily.

tk_focusNext()
Return the next widget in the focus order which follows widget which has currently the focus.

The focus order first goes to the next child, then to the children of the child recursively and then to the next
sibling which is higher in the stacking order. A widget is omitted if it has the takefocus resource set to 0.

tk_focusPrev()
Return previous widget in the focus order. See tk_focusNext for details.

tk_menuBar(*args)
Do not use. Needed in Tk 3.6 and earlier.

tk_setPalette(*args, **kw)
Set a new color scheme for all widget elements.

A single color as argument will cause that all colors of Tk widget elements are derived from this. Alternatively several keyword parameters and its associated colors can be given. The following keywords are valid: activeBackgroundColor, foreground, selectColor, activeForegroundColor, highlightBackgroundColor, selectBackgroundColor, background, highlightColor, selectForeground, disabledForeground, insertBackgroundColor, troughColor.

tk_strictMotif(boolean=None)
Set Tcl internal variable, whether the look and feel should adhere to Motif.

A parameter of 1 means adhere to Motif (e.g. no color change if mouse passes over slider). Returns the set value.

tkraise(aboveThis=None)
Raise this widget in the stacking order.

transient(master=None)
Instruct the window manager that this widget is transient with regard to widget MASTER.

unbind(sequence, funcid=None)
Unbind for this widget for event SEQUENCE the function identified with FUNCID.

unbind_all(sequence)
Unbind for all widgets for event SEQUENCE all functions.

unbind_class(className, sequence)
Unbind for a all widgets with bindtag CLASSNAME for event SEQUENCE all functions.

update()
Enter event loop until all pending events have been processed by Tcl.

update_idletasks()
Enter event loop until all idle callbacks have been called. This will update the display of windows but not process events caused by the user.

wait_variable(name='PY_VAR')
Wait until the variable is modified.

A parameter of type IntVar, StringVar, DoubleVar or BooleanVar must be given.

wait_visibility(window=None)
Wait until the visibility of a WIDGET changes (e.g. it appears).

If no parameter is given self is used.

wait_window(window=None)
Wait until a WIDGET is destroyed.
If no parameter is given self is used.

```
waitvar (name='PY_VAR')
```

Wait until the variable is modified.

A parameter of type IntVar, StringVar, DoubleVar or BooleanVar must be given.

```
waitvar
```

If no parameter is given self is used.

```
waitvar (name='PY_VAR')
```

Wait until the variable is modified.

A parameter of type IntVar, StringVar, DoubleVar or BooleanVar must be given.

```
waitvar
```

If no parameter is given self is used.

```
waitvar (name='PY_VAR')
```

Wait until the variable is modified.

A parameter of type IntVar, StringVar, DoubleVar or BooleanVar must be given.

```
waitvar
```

If no parameter is given self is used.

```
waitvar (name='PY_VAR')
```

Wait until the variable is modified.

A parameter of type IntVar, StringVar, DoubleVar or BooleanVar must be given.

```
waitvar
```

If no parameter is given self is used.

```
waitvar (name='PY_VAR')
```

Wait until the variable is modified.

A parameter of type IntVar, StringVar, DoubleVar or BooleanVar must be given.

```
waitvar
```

If no parameter is given self is used.

```
waitvar (name='PY_VAR')
```

Wait until the variable is modified.

A parameter of type IntVar, StringVar, DoubleVar or BooleanVar must be given.

```
waitvar
```

If no parameter is given self is used.

```
waitvar (name='PY_VAR')
```

Wait until the variable is modified.

A parameter of type IntVar, StringVar, DoubleVar or BooleanVar must be given.

```
waitvar
```

If no parameter is given self is used.

```
waitvar (name='PY_VAR')
```

Wait until the variable is modified.

A parameter of type IntVar, StringVar, DoubleVar or BooleanVar must be given.
```

winfo_pixels(number)
Rounded integer value of winfo_fpixels.

winfo_pointerx()
Return the x coordinate of the pointer on the root window.

winfo_pointerxy()
Return a tuple of x and y coordinates of the pointer on the root window.

winfo_pointery()
Return the y coordinate of the pointer on the root window.

winfo_reqheight()
Return requested height of this widget.

winfo_reqwidth()
Return requested width of this widget.

winfo_rgb(color)
Return tuple of decimal values for red, green, blue for COLOR in this widget.

winfo_rootx()
Return x coordinate of upper left corner of this widget on the root window.

winfo_rooty()
Return y coordinate of upper left corner of this widget on the root window.

winfo_screen()
Return the screen name of this widget.

winfo_screencells()
Return the number of the cells in the colormap of the screen of this widget.

winfo_screendepth()
Return the number of bits per pixel of the root window of the screen of this widget.

winfo_screenheight()
Return the number of pixels of the height of the screen of this widget in pixel.

winfo_screenmmheight()
Return the number of pixels of the height of the screen of this widget in mm.

winfo_screenmmwidth()
Return the number of pixels of the width of the screen of this widget in mm.

winfo_screenvisual()
Return one of the strings directcolor, grayscale, pseudocolor, staticcolor, staticgray, or truecolor for the
default colormodel of this screen.

winfo_screenwidth()
Return the number of pixels of the width of the screen of this widget in pixel.

winfo_server()
Return information of the X-Server of the screen of this widget in the form “XmajorRminor vendor vendorVersion”.

winfo_toplevel()
Return the toplevel widget of this widget.

winfo_viewable()
Return true if the widget and all its higher ancestors are mapped.
```
winfo_visual()  
Return one of the strings directcolor, grayscale, pseudocolor, staticcolor, staticgray, or truecolor for the color model of this widget.

winfo_visualid()  
Return the X identifier for the visual for this widget.

winfo_visualsavailable(includeids=0)  
Return a list of all visuals available for the screen of this widget.

Each item in the list consists of a visual name (see winfo_visual), a depth and if INCLUDEIDS=1 is given also the X identifier.

winfo_vrootheight()  
Return the height of the virtual root window associated with this widget in pixels. If there is no virtual root window return the height of the screen.

winfo_vrootwidth()  
Return the width of the virtual root window associated with this widget in pixels. If there is no virtual root window return the width of the screen.

winfo_vrootx()  
Return the x offset of the virtual root relative to the root window of the screen of this widget.

winfo_vrooty()  
Return the y offset of the virtual root relative to the root window of the screen of this widget.

winfo_width()  
Return the width of this widget.

winfo_x()  
Return the x coordinate of the upper left corner of this widget in the parent.

winfo_y()  
Return the y coordinate of the upper left corner of this widget in the parent.

withdraw()  
Withdraw this widget from the screen such that it is unmapped and forgotten by the window manager. Re-draw it with wm_deiconify.

wm_aspect(minNumer=None, minDenom=None, maxNumer=None, maxDenom=None)  
Instruct the window manager to set the aspect ratio (width/height) of this widget to be between MINNUMER/MINDENOM and MAXNUMER/MAXDENOM. Return a tuple of the actual values if no argument is given.

wm_attributes(*args)  
This subcommand returns or sets platform specific attributes

The first form returns a list of the platform specific flags and their values. The second form returns the value for the specific option. The third form sets one or more of the values. The values are as follows:

On Windows, -disabled gets or sets whether the window is in a disabled state. -toolwindow gets or sets the style of the window to toolwindow (as defined in the MSDN). -topmost gets or sets whether this is a topmost window (displays above all other windows).

On Macintosh, XXXXX

On Unix, there are currently no special attribute values.

wm_client(name=None)  
Store NAME in WM_CLIENT_MACHINE property of this widget. Return current value.
Robot Framework Documentation, Release 2.7.7

`wm_colormapwindows(*wlist)`
Store list of window names (WLIST) into WM_COLORMAPWINDOWS property of this widget. This list contains windows whose colormaps differ from their parents. Return current list of widgets if WLIST is empty.

`wm_command(value=None)`
Store VALUE in WM_COMMAND property. It is the command which shall be used to invoke the application. Return current command if VALUE is None.

`wm_deiconify()`
Deiconify this widget. If it was never mapped it will not be mapped. On Windows it will raise this widget and give it the focus.

`wm_focusmodel(model=None)`
Set focus model to MODEL. “active” means that this widget will claim the focus itself, “passive” means that the window manager shall give the focus. Return current focus model if MODEL is None.

`wm_frame()`
Return identifier for decorative frame of this widget if present.

`wm_geometry(newGeometry=None)`
Set geometry to NEWGEOMETRY of the form =widthxheight+x+y. Return current value if None is given.

`wm_grid(baseWidth=None, baseHeight=None, widthInc=None, heightInc=None)`
Instruct the window manager that this widget shall only be resized on grid boundaries. WIDTHINC and HEIGHTINC are the width and height of a grid unit in pixels. BASEWIDTH and BASEHEIGHT are the number of grid units requested in Tk_GeometryRequest.

`wm_group(pathName=None)`
Set the group leader widgets for related widgets to PATHNAME. Return the group leader of this widget if None is given.

`wm_iconbitmap(bitmap=None, default=None)`
Set bitmap for the iconified widget to BITMAP. Return the bitmap if None is given.

Under Windows, the DEFAULT parameter can be used to set the icon for the widget and any descendents that don’t have an icon set explicitly. DEFAULT can be the relative path to a .ico file (example: root.iconbitmap(default=’myicon.ico’)). See Tk documentation for more information.

`wm_iconify()`
Display widget as icon.

`wm_iconmask(bitmap=None)`
Set mask for the icon bitmap of this widget. Return the mask if None is given.

`wm_iconname(newName=None)`
Set the name of the icon for this widget. Return the name if None is given.

`wm_iconposition(x=None, y=None)`
Set the position of the icon of this widget to X and Y. Return a tuple of the current values of X and X if None is given.

`wm_iconwindow(pathName=None)`
Set widget PATHNAME to be displayed instead of icon. Return the current value if None is given.

`wm_maxsize(width=None, height=None)`
Set max WIDTH and HEIGHT for this widget. If the window is gridded the values are given in grid units. Return the current values if None is given.

`wm_minsize(width=None, height=None)`
Set min WIDTH and HEIGHT for this widget. If the window is gridded the values are given in grid units. Return the current values if None is given.
**wm_overrideredirect** *(boolean=None)*

Instruct the window manager to ignore this widget if BOOLEAN is given with 1. Return the current value if None is given.

**wm_positionfrom** *(who=None)*

Instruct the window manager that the position of this widget shall be defined by the user if WHO is “user”, and by its own policy if WHO is “program”.

**wm_protocol** *(name=None, func=None)*

Bind function FUNC to command NAME for this widget. Return the function bound to NAME if None is given. NAME could be e.g. “WM_SAVE_YOURSELF” or “WM_DELETE_WINDOW”.

**wm_resizable** *(width=None, height=None)*

Instruct the window manager whether this width can be resized in WIDTH or HEIGHT. Both values are boolean values.

**wm_sizefrom** *(who=None)*

Instruct the window manager that the size of this widget shall be defined by the user if WHO is “user”, and by its own policy if WHO is “program”.

**wm_state** *(newstate=None)*

Query or set the state of this widget as one of normal, icon, iconic (see wm_iconwindow), withdrawn, or zoomed (Windows only).

**wm_title** *(string=None)*

Set the title of this widget.

**wm_transient** *(master=None)*

Instruct the window manager that this widget is transient with regard to widget MASTER.

**wm_withdraw** *

Withdraw this widget from the screen such that it is unmapped and forgotten by the window manager. Re-draw it with wm_deiconify.

### 4.8 model Package

#### 4.8.1 model Package

Contains base classes and other generic functionality.

In RF 2.7 this package is mainly used by `robot.result` package, but there is a plan to change also `robot.running` to use this in RF 2.8.

This package is considered stable.

#### 4.8.2 criticality Module

**class** `robot.model.criticality.Criticality` *(critical_tags=None, non_critical_tags=None)*

Bases: `object`

- `tag_is_critical` *(tag)*
- `tag_is_non_critical` *(tag)*
- `test_is_critical` *(test)*
4.8.3 filter Module

class robot.model.filter.Filter(include_suites=None, include_tests=None, include_tags=None, exclude_tags=None):
    Bases: robot.model.visitor.SuiteVisitor
        include_suites
        include_tests
        include_tags
        exclude_tags
        start_suite(suite)
        end_keyword(keyword)
        end_message(msg)
        end_suite(suite)
        end_test(test)
        start_keyword(keyword)
        start_message(msg)
        start_test(test)
        visit_message(msg)
        visit_suite(suite)
        visit_test(test)
        visit_keyword(keyword)

4.8.4 itemlist Module

class robot.model.itemlist.ItemList(item_class, common_attrs=None, items=None):
    Bases: object
        create(*args, **kwargs)
        append(item)
        extend(items)
        index(item)
        clear()
        visit(visitor)

4.8.5 keyword Module

class robot.model.keyword.Keyword(name='', doc='', args=None, type='kw', timeout=' '):
    Bases: robot.model.modelobject.ModelObject
    KEYWORD_TYPE = 'kw'
    SETUP_TYPE = 'setup'
    TEARDOWN_TYPE = 'teardown'
FOR_LOOP_TYPE = 'for'
FOR_ITEM_TYPE = 'foritem'
message_class
    alias of Message
parent
name
doc
args
type
timeout
keywords
messages
id
visit(visitor)
class robot.model.keyword.Keywords(keyword_class=<class 'robot.model.keyword.Keyword'>,
    parent=None, keywords=None)
    Bases: robot.model.itemlist.ItemList
setup
tear down
all
append(item)
clear()
create(*args, **kwargs)
extend(items)
index(item)
visit(visitor)
normal

4.8.6 message Module

class robot.model.message.Message(message='', level='INFO', html=False, timestamp=None, parent=None)
    Bases: robot.model.modelobject.ModelObject
message
level
html
timestamp
parent
html_message
visit(visitor)

class robot.model.message.Messages(message_class=<class 'robot.model.message.Message'>, parent=None, messages=None)
    Bases: robot.model.itemlist.ItemList
    append(item)
    clear()
    create(*args, **kwargs)
    extend(items)
    index(item)
    visit(visitor)

4.8.7 metadata Module

class robot.model.metadata.Metadata(initial=None)
    Bases: robot.utils.normalizing.NormalizedDict
    clear()
    copy()
    classmethod fromkeys(iterable, value=None)
    get(key, default=None)
    has_key(key)
    items()
    iteritems()
    iterkeys()
    itervalues()
    keys()
    pop(key)
    popitem()
    set(key, value)
   .setdefault(key=failobj=None)
    update(dict=None, **kwargs)
    values()

4.8.8 modelobject Module

class robot.model.modelobject.ModelObject
    Bases: object
4.8.9 namepatterns Module

```python
class robot.model.namepatterns.SuiteNamePatterns(patterns=None)
    Bases: robot.model.namepatterns._NamePatterns
    match(name, longname=None)

class robot.model.namepatterns.TestNamePatterns(patterns=None)
    Bases: robot.model.namepatterns._NamePatterns
    match(name, longname=None)
```

4.8.10 statistics Module

```python
class robot.model.statistics.Statistics(suite, suite_level=-1, tag_stat_include=None, tag_stat_exclude=None, tag_stat_combine=None, tag_doc=None, tag_stat_link=None)
    Bases: object
    visit(visitor)

class robot.model.statistics.StatisticsBuilder(total_builder, suite_builder, tag_builder)
    Bases: robot.model.visitor.SuiteVisitor
    start_suite(suite)
    end_suite(suite)
    visit_test(test)
    visit_keyword(kw)
    end_keyword(keyword)
    end_message(msg)
    end_test(test)
    start_keyword(keyword)
    start_message(msg)
    start_test(test)
    visit_message(msg)
    visit_suite(suite)
```

4.8.11 stats Module

```python
class robot.model.stats.Stat(name)
    Bases: object
    get_attributes(include_label=False, include_elapsed=False, exclude_empty=False, values_as_strings=False, html_escape=False)
    total
    add_test(test)
    visit(visitor)

class robot.model.stats.TotalStat(name)
    Bases: robot.model.stats.Stat
```
```python
type = 'total'
add_test(test)

get_attributes(include_label=False, include_elapsed=False, exclude_empty=False, values_as_strings=False, html_escape=False)

total
visit(visitor)
class robot.model.stats.SuiteStat(suite)
    Bases: robot.model.stats.Stat
    type = 'suite'
    add_stat(other)
    add_test(test)
    get_attributes(include_label=False, include_elapsed=False, exclude_empty=False, values_as_strings=False, html_escape=False)

total
visit(visitor)
class robot.model.stats.TagStat(name, doc='', links=None, critical=False, non_critical=False, combined='')
    Bases: robot.model.stats.Stat
    type = 'tag'
    info
    add_test(test)
    get_attributes(include_label=False, include_elapsed=False, exclude_empty=False, values_as_strings=False, html_escape=False)

total
visit(visitor)
class robot.model.stats.CombinedTagStat(pattern, name=None, doc='', links=None)
    Bases: robot.model.stats.TagStat
    match(tags)
    add_test(test)
    get_attributes(include_label=False, include_elapsed=False, exclude_empty=False, values_as_strings=False, html_escape=False)

info
total

4.8.12 suitestatistics Module

class robot.model.suitestatistics.SuiteStatistics(suite)
    Bases: object
    visit(visitor)
```

4.8. model Package 377
class robot.model.suitestatistics.SuiteStatisticsBuilder (suite_stat_level)
    Bases: object
    current
    start_suite (suite)
    add_test (test)
    end_suite ()

4.8.13 tags Module

class robot.model.tags.Tags (tags=None)
    Bases: object
    add (tags)
    remove (tags)
    match (tags)

class robot.model.tags.TagPatterns (patterns)
    Bases: object
    match (tags)

robot.model.tags.TagPattern (pattern)

4.8.14 tagsetter Module

class robot.model.tagsetter.TagSetter (add=None, remove=None)
    Bases: robot.model.visitor.SuiteVisitor
    start_suite (suite)
    visit_test (test)
    visit_keyword (keyword)
    end_keyword (keyword)
    end_message (msg)
    end_suite (suite)
    end_test (test)
    start_keyword (keyword)
    start_message (msg)
    start_test (test)
    visit_message (msg)
    visit_suite (suite)
4.8.15 tagstatistics Module

class robot.model.tagstatistics.TagStatistics(combined_stats)
    Bases: object
    visit(visitor)

class robot.model.tagstatistics.TagStatisticsBuilder(criticality=None, included=None, excluded=None, combined=None, docs=None, links=None)
    Bases: object
    add_test(test)

class robot.model.tagstatistics.TagStatInfo(criticality=None, docs=None, links=None)
    Bases: object
    get_stat(tag)
    get_combined_stats(combined=None)
    get_combined_stat(pattern, name=None)
    get_doc(tag)
    get_links(tag)

class robot.model.tagstatistics.TagStatDoc(pattern, doc)
    Bases: object
    match(tag)

class robot.model.tagstatistics.TagStatLink(pattern, link, title)
    Bases: object
    match(tag)
    get_link(tag)

4.8.16 testcase Module

class robot.model.testcase.TestCase(name='', doc='', tags=None, timeout='')
    Bases: robot.model.modelobject.ModelObject
    keyword_class
        alias of Keyword

    parent
    name
    doc
    timeout
    tags
    keywords
    id
    longname
    critical
    visit(visitor)
class robot.model.testcase.TestCases(test_class=<class 'robot.model.testcase.TestCase'>, parent=None, tests=None)
Bases: robot.model.itemlist.ItemList
    append(item)
    clear()
    create(*args, **kwargs)
    extend(items)
    index(item)
    visit(visitor)

4.8.17 testsuite Module

class robot.model.testsuite.TestSuite(source='', name='', doc='', metadata=None)
Bases: robot.model.modelobject.ModelObject
    test_class
        alias of TestCase
    keyword_class
        alias of Keyword
    parent
    source
    doc
    name
    set_criticality(critical_tags=None, non_critical_tags=None)
    criticality
    metadata
    suites
    tests
    keywords
    id
    longname
    test_count
    set_tags(add=None, remove=None)
    filter(included_suites=None, included_tests=None, included_tags=None, excluded_tags=None)
    visit(visitor)

class robot.model.testsuite.TestSuites(suite_class=<class 'robot.model.testsuite.TestSuite'>, parent=None, suites=None)
Bases: robot.model.itemlist.ItemList
    append(item)
    clear()
create(*args, **kwargs)
extend(items)
index(item)
visit(visitor)

### 4.8.18 totalstatistics Module

```python
class TotalStatistics(robot.model.totalstatistics.TotalStatistics):
    Bases: object

    def visit(self, visitor):
        pass

    def message(self):
        pass

class TotalStatisticsBuilder(robot.model.totalstatistics.TotalStatisticsBuilder):
    Bases: robot.model.visitor.SuiteVisitor

    def add_test(self, test):
        pass

    def visit_test(self, test):
        pass

    def visit_keyword(self, kw):
        pass

    def end_keyword(self, keyword):
        pass

    def end_message(self, msg):
        pass

    def end_suite(self, suite):
        pass

    def end_test(self, test):
        pass

    def start_keyword(self, keyword):
        pass

    def start_message(self, msg):
        pass

    def start_suite(self, suite):
        pass

    def start_test(self, test):
        pass

    def visit_message(self, msg):
        pass

    def visit_suite(self, suite):
        pass
```

### 4.8.19 visitor Module

```python
class SuiteVisitor(robot.model.visitor.SuiteVisitor):
    Bases: object

    def visit_suite(self, suite):
        pass

    def start_suite(self, suite):
        pass

    def end_suite(self, suite):
        pass

    def visit_test(self, test):
        pass

    def start_test(self, test):
        pass

    def end_test(self, test):
        pass

    def visit_keyword(self, kw):
        pass

    def start_keyword(self, keyword):
        pass
```

4.8. model Package
```python
end_keyword(keyword)
visit_message(msg)
start_message(msg)
end_message(msg)
class robot.model.visitor.SkipAllVisitor
    Bases: robot.model.visitor.SuiteVisitor
    visit_suite(suite)
    visit_keyword(kw)
    visit_test(test)
    visit_message(msg)
    end_keyword(keyword)
    end_message(msg)
    end_suite(suite)
    end_test(test)
    start_keyword(keyword)
    start_message(msg)
    start_suite(suite)
    start_test(test)
```
start_keyword (kw)
end_keyword (kw)
output_file (name, path)
close()
debug (msg)
error (msg)
fail (msg)
info (msg)
set_level (level)
trace (msg)
warn (msg)
write (message, level, html=False)

4.9.4 highlighting Module

robot.output.highlighting.Highlighter (stream)
class robot.output.highlighting.AnsiHighlighter (stream)
    Bases: object
    green()
    red()
    yellow()
    reset()

class robot.output.highlighting.NoHighlighting (stream)
    Bases: robot.output.highlighting.AnsiHighlighter
    green()
    red()
    reset()
    yellow()

class robot.output.highlighting.DosHighlighter (stream)
    Bases: object
    green()
    red()
    yellow()
    reset()
4.9.5 listeners Module

```python
class robot.output.listeners.Listeners(listeners):
    Bases: object
    start_suite(*args)
    end_suite(*args)
    start_test(*args)
    end_test(*args)
    start_keyword(*args)
    end_keyword(*args)
    log_message(*args)
    message(*args)
    output_file(*args)
    close(*args)
```

4.9.6 logger Module

```python
class robot.output.logger.Logger
    Bases: robot.output.loggerhelper.AbstractLogger
    A global logger proxy to which new loggers may be registered.
    Whenever something is written to LOGGER in code, all registered loggers are notified. Messages are also cached and cached messages written to new loggers when they are registered.
    Tools using Robot Framework’s internal modules should register their own loggers at least to get notifications about errors and warnings. A shortcut to get errors/warnings into console is using 'register_console_logger'.

    disable_message_cache()
    disable_automatic_console_logger()
    register_logger(*loggers)
    register_context_changing_logger(logger)
    unregister_logger(*loggers)
    register_console_logger(width=78, colors='AUTO', markers='AUTO', stdout=None, stderr=None)
    register_file_logger(path=None, level='INFO')
    message(msg)
        Messages about what the framework is doing, warnings, errors, ...
    log_message(msg)
        Messages about what the framework is doing, warnings, errors, ...
    log_output(output)
    enable_library_import_logging()
    disable_library_import_logging()
```
**output_file** *(name, path)*

Finished output, report, log, debug, or xunit file

```python
close()
start_suite(suite)
end_suite(suite)
start_test(test)
end_test(test)
start_keyword(keyword)
end_keyword(keyword)
debug(msg)
error(msg)
fail(msg)
info(msg)
set_level(level)
trace(msg)
warn(msg)
write(message, level, html=False)
class robot.output.logger.LoggerCollection
    Bases: object
    register_regular_logger(logger)
    register_context_changing_logger(logger)
    remove_first_regular_logger()
    unregister_logger(logger)
    starting_loggers()
    ending_loggers()
    all_loggers()
```

### 4.9.7 loggerhelper Module

class robot.output.loggerhelper(AbstractLogger(level='TRACE'))

```python
    set_level(level)
    trace(msg)
    debug(msg)
    info(msg)
    warn(msg)
    fail(msg)
    error(msg)
```
write (message, level, html=False)

message (msg)

class robot.output.loggerhelper.Message (message, level='INFO', html=False, timestamp=None)
    Bases: robot.model.message.Message
timestamp
message
html
html_message
level
parent
visit (visitor)

class robot.output.loggerhelper.IsLogged (level)

    set_level (level)

class robot.output.loggerhelper.AbstractLoggerProxy (logger)

4.9.8 monitor Module

class robot.output.monitor.CommandLineMonitor (width=78, colors='AUTO', markers='AUTO', stdout=None, stderr=None)
    Bases: object
    start_suite (suite)
    end_suite (suite)
    start_test (test)
    end_test (test)
    start_keyword (kw)
    end_keyword (kw)
    message (msg)
    output_file (name, path)

class robot.output.monitor.CommandLineWriter (width=78, colors='AUTO', markers='AUTO', stdout=None, stderr=None)
    Bases: object
    info (name, doc, start_suite=False)
    suite_separator ()
    test_separator ()
    status (status, clear=False)
    message (message)
    keyword_marker (kw)
    error (message, level, clear=False)
output (name, path)

class robot.output.monitor.StatusHighlighter (colors, *streams)
    Bases: object
    highlight_status (status, stream)
    highlight (text, color, stream)

class robot.output.monitor.KeywordMarker (markers, stdout, highlighter)
    Bases: object
    mark (kw)
    reset_count ()

4.9.9 output Module

class robot.output.output.Output (settings)
    Bases: robot.output.loggerhelper.AbstractLogger
    close (suite)
    start_suite (suite)
    end_suite (suite)
    start_test (test)
    end_test (test)
    start_keyword (kw)
    end_keyword (kw)
    message (msg)
    set_log_level (level)
    debug (msg)
    error (msg)
    fail (msg)
    info (msg)
    set_level (level)
    trace (msg)
    warn (msg)
    write (message, level, html=False)

4.9.10 pyloggingconf Module

robot.output.pyloggingconf.initialize (level)
robot.output.pyloggingconf.set_level (level)

class robot.output.pyloggingconf.RobotHandler (level=0)
    Bases: logging.Handler
    Initializes the instance - basically setting the formatter to None and the filter list to empty.

4.9. output Package
emit (record)

acquire()
    Acquire the I/O thread lock.

addFilter (filter)
    Add the specified filter to this handler.

close()
    Tidy up any resources used by the handler.
    This version removes the handler from an internal map of handlers, _handlers, which is used for handler lookup by name. Subclasses should ensure that this gets called from overridden close() methods.

createLock()
    Acquire a thread lock for serializing access to the underlying I/O.

filter (record)
    Determine if a record is loggable by consulting all the filters.
    The default is to allow the record to be logged; any filter can veto this and the record is then dropped. Returns a zero value if a record is to be dropped, else non-zero.

flush()
    Ensure all logging output has been flushed.
    This version does nothing and is intended to be implemented by subclasses.

format (record)
    Format the specified record.
    If a formatter is set, use it. Otherwise, use the default formatter for the module.

get_name()

handle (record)
    Conditionally emit the specified logging record.
    Emission depends on filters which may have been added to the handler. Wrap the actual emission of the record with acquisition/release of the I/O thread lock. Returns whether the filter passed the record for emission.

handleError (record)
    Handle errors which occur during an emit() call.
    This method should be called from handlers when an exception is encountered during an emit() call. If raiseExceptions is false, exceptions get silently ignored. This is what is mostly wanted for a logging system - most users will not care about errors in the logging system, they are more interested in application errors. You could, however, replace this with a custom handler if you wish. The record which was being processed is passed in to this method.

name

release()
    Release the I/O thread lock.

removeFilter (filter)
    Remove the specified filter from this handler.

setFormatter (fmt)
    Set the formatter for this handler.

setLevel (level)
    Set the logging level of this handler.
set_name(name)

4.9.11 stdoutlogsplitter Module

class robot.output.stdoutlogsplitter.StdoutLogSplitter(output)
    Bases: object
    Splits messages logged through stdout (or stderr) into Message objects

4.9.12 xmllogger Module

class robot.output.xmllogger.XmlLogger(path, log_level='TRACE', generator='Robot')
    Bases: object
    close()
    set_log_level(level)
    message(msg)
    log_message(msg)
    start_keyword(kw)
    end_keyword(kw)
    start_test(test)
    end_test(test)
    start_suite(suite)
    end_suite(suite)
    start_statistics(stats)
    end_statistics(stats)
    start_total_stats(total_stats)
    end_total_stats(total_stats)
    start_tag_stats(tag_stats)
    end_tag_stats(tag_stats)
    start_suite_stats(tag_stats)
    end_suite_stats(tag_stats)
    total_stat(stat)
    suite_stat(stat)
    tag_stat(stat)
    start_errors()
    end_errors()
4.10 parsing Package

4.10.1 parsing Package

Implements parsing of test data files.

Classes `TestCaseFile`, `TestDataDirectory` and `ResourceFile` represented parsed test data. These can be modified and saved back to disk. In addition, a convenience factory function `TestData()` can be used to parse file or directory to a corresponding object.

This package is considered stable.

Example

```python
from robot.parsing import TestCaseFile

suite = TestCaseFile(source='path/to/tests.html').populate()
print('Suite: ', suite.name)
for test in suite.testcase_table:
    print(test.name)

robot.parsing.disable_curdir_processing(method)
    # Decorator to disable processing ${CURDIR} variable.
```

4.10.2 comments Module

```python
class robot.parsing.comments.CommentCache
    Bases: object
    add(comment)
    consume_with(function)

class robot.parsing.comments.Comments
    Bases: object
    add(row)
    value

class robot.parsing.comments.Comment(comment_data)
    Bases: object
    as_list()
```

4.10.3 datarow Module

```python
class robot.parsing.datarow.DataRow(cells)
    Bases: object
    head
    all
    data
dedent()
```
handle_old_style_metadata()
starts_for_loop()
starts_test_or_user_keyword_setting()
test_or_user_keyword_setting_name()
is_indented()
is_continuing()
is_commented()

4.10.4 htmlreader Module

class robot.parsing.htmlreader.HtmlReader
    Bases: HTMLParser.HTMLParser

    IGNORE = 0
    INITIAL = 1
    PROCESS = 2
    read(htmlfile, populator)
    handle_starttag(tag, attrs)
    handle_endtag(tag)
    handle_data(data)
    handle_entityref(name)
    handle_charref(number)
    handle_pi(data)
    unknown_decl(data)
    table_start(attrs=None)
    table_end()
    tr_start(attrs=None)
    tr_end()
    td_start(attrs=None)
    td_end()
    br_start(attrs=None)
    meta_start(attrs)

    CDATA_CONTENT_ELEMENTS = ('script', 'style')
    check_for_whole_start_tag(i)
    clear_cdata_mode()
    close()
        Handle any buffered data.
    entitydefs = None
    error(message)
feed(data)
Feed data to the parser.
Call this as often as you want, with as little or as much text as you want (may include ‘n’).

get_starttag_text()
Return full source of start tag: ‘<...>’.

getpos()
Return current line number and offset.
goahead(end)
handle_comment(data)
handle_decl(decl)
handle_startendtag(tag, attrs)
parse_bogus_comment(i, report=1)
parse_comment(i, report=1)
parse_declaration(i)
parse_endtag(i)
parse_html_declaration(i)
parse_marked_section(i, report=1)
parse_pi(i)
parse_starttag(i)
reset()
Reset this instance. Loses all unprocessed data.
set_cdata_mode(elem)
unescape(s)
updatepos(i, j)

4.10.5 model Module

robot.parsing.model.TestData(parent=None, source=None, include_suites=[], warn_on_skipped=False)
Parses a file or directory to a corresponding model object.

Parameters
• parent – (optional) parent to be used in creation of the model object.
• source – path where test data is read from.

Returns TestDataDirectory if source is a directory, TestCaseFile otherwise.

class robot.parsing.model.TestCaseFile(parent=None, source=None)
Bases: robot.parsing.model._TestData
The parsed test case file object.

populate()
has_tests()
Robot Framework Documentation, Release 2.7.7

```
import keywords

name

report_invalid_syntax(table, message, level='ERROR')

save(**options)
    Writes this datafile to disk.

    Parameters options -- Configuration for writing. These are passed to WritingContext as
    keyword arguments.

    See also robot.writer.datafilewriter.DataFileWriter

start_table(header_row)

class robot.parsing.model.ResourceFile(source=None)
    Bases: robot.parsing.model._TestData
    The parsed resource file object.

    populate()  
    imports
    keywords
    name

report_invalid_syntax(table, message, level='ERROR')

save(**options)
    Writes this datafile to disk.

    Parameters options -- Configuration for writing. These are passed to WritingContext as
    keyword arguments.

    See also robot.writer.datafilewriter.DataFileWriter

start_table(header_row)

class robot.parsing.model.TestDataDirectory(parent=None, source=None)
    Bases: robot.parsing.model._TestData
    The parsed test data directory object. Contains hierarchical structure of other TestDataDirectory
    and TestCaseFile objects.

    populate(include_suites=[], warn_on_skipped=False, recurse=True)

    add_child(path, include_suites)

    has_tests()
    imports
    keywords
    name

report_invalid_syntax(table, message, level='ERROR')

save(**options)
    Writes this datafile to disk.

    Parameters options -- Configuration for writing. These are passed to WritingContext as
    keyword arguments.
```
See also `robot.writer.datafilewriter.DataFileWriter`

```python
start_table (header_row)
```

class `robot.parsing.model.TestCaseFileSettingTable` (parent)
  Bases: `robot.parsing.model._SettingTable`
  
  add_library (name, args=None, comment=None)
  add_metadata (name, value='', comment=None)
  add_resource (name, invalid_args=None, comment=None)
  add_variables (name, args=None, comment=None)

  directory
  get_setter (setting_name)
  header
  is_setting (setting_name)
  name
  normalize (setting)
  report_invalid_syntax (message, level='ERROR')
  set_header (header)

  source
type = 'setting'
```

class `robot.parsing.model.ResourceFileSettingTable` (parent)
  Bases: `robot.parsing.model._SettingTable`
  
  add_library (name, args=None, comment=None)
  add_metadata (name, value='', comment=None)
  add_resource (name, invalid_args=None, comment=None)
  add_variables (name, args=None, comment=None)

  directory
  get_setter (setting_name)
  header
  is_setting (setting_name)
  name
  normalize (setting)
  report_invalid_syntax (message, level='ERROR')
  set_header (header)

  source
type = 'setting'
```

class `robot.parsing.model.InitFileSettingTable` (parent)
  Bases: `robot.parsing.model._SettingTable`
  
  add_library (name, args=None, comment=None)
add_metadata (name, value='', comment=None)
add_resource (name, invalid_args=None, comment=None)
add_variables (name, args=None, comment=None)
directory
get_setter (setting_name)
header
is_setting (setting_name)
name
normalize (setting)
report_invalid_syntax (message, level='ERROR')
set_header (header)
source
type = 'setting'
class robot.parsing.model.VariableTable (parent)
   Bases: robot.parsing.model._Table
type = 'variable'
add (name, value, comment=None)
directory
header
name
report_invalid_syntax (message, level='ERROR')
set_header (header)
source
class robot.parsing.model.TestCaseTable (parent)
   Bases: robot.parsing.model._Table
type = 'test case'
add (name)
is_started ()
directory
header
name
report_invalid_syntax (message, level='ERROR')
set_header (header)
source
class robot.parsing.model.KeywordTable (parent)
   Bases: robot.parsing.model._Table
type = 'keyword'
add(name)
directory
header
name
report_invalid_syntax(message, level='ERROR')
set_header(header)
source
class robot.parsing.model.Variable(name, value, comment=None)
    Bases: object
    as_list()
is_set()
is_for_loop()
has_data()
class robot.parsing.model.TestCase(parent, name)
    Bases: robot.parsing.model._WithSteps, robot.parsing.model._WithSettings
    source
directory
add_for_loop(declaration, comment=None)
report_invalid_syntax(message, level='ERROR')
settings
add_step(content, comment=None)
copy(name)
get_setter(setting_name)
is_setting(setting_name)
normalize(setting)
class robot.parsing.model.UserKeyword(parent, name)
    Bases: robot.parsing.model.TestCase
    settings
    add_for_loop(declaration, comment=None)
    add_step(content, comment=None)
copy(name)
directory
get_setter(setting_name)
is_setting(setting_name)
normalize(setting)
report_invalid_syntax(message, level='ERROR')
source
class `robot.parsing.model.ForLoop`(declaration, comment=None)
Bases: `robot.parsing.model._WithSteps`

- `is_comment()`
- `is_for_loop()`
- `apply_template(template)`
- `as_list(indent=False, include_comment=True)`
- `is_set()`
- `add_step(content, comment=None)`
- `copy(name)`

class `robot.parsing.model.Step`(content, comment=None)
Bases: object

- `is_comment()`
- `is_for_loop()`
- `apply_template(template)`
- `is_set()`
- `as_list(indent=False, include_comment=True)`

class `robot.parsing.model.OldStyleSettingAndVariableTableHeaderMatcher`
Bases: object

- `match(header)`

class `robot.parsing.model.OldStyleTestAndKeywordTableHeaderMatcher`
Bases: object

- `match(header)`

### 4.10.6 populators Module

class `robot.parsing.populators.FromFilePopulator`(datafile)
Bases: object

- `populate(path)`
- `start_table(header)`
- `eof()`
- `add(row)`

class `robot.parsing.populators.FromDirectoryPopulator`
Bases: object

- `ignored_prefixes = ('_', '.')`
- `ignored_dirs = ('CVS',)`
- `populate(path, datadir, include_suites, warn_on_skipped, recurse=True)`

### 4.10.7 restreader Module

`robot.parsing.restreader.RestReader()`

4.10. parsing Package
4.10.8 settings Module

class robot.parsing.settings.Setting (setting_name, parent=None, comment=None)
Bases: object

   reset ()
   source
directory

   populate (value, comment=None)
       Mainly used at parsing time, later attributes can be set directly.

   is_set ()
   is_for_loop ()

   report_invalid_syntax (message, level='ERROR')

   as_list ()

class robot.parsing.settings.StringValueJoiner (separator)
Bases: object

   join_string_with_value (string, value)

   string_value (value)

class robot.parsing.settings.Documentation (setting_name, parent=None, comment=None)
Bases: robot.parsing.settings.Setting

   as_list ()
directory

   is_for_loop ()
   is_set ()

   populate (value, comment=None)
       Mainly used at parsing time, later attributes can be set directly.

   report_invalid_syntax (message, level='ERROR')

   reset ()

   source

class robot.parsing.settings.Template (setting_name, parent=None, comment=None)
Bases: robot.parsing.settings.Setting

   is_set ()
   as_list ()
directory

   is_for_loop ()

   populate (value, comment=None)
       Mainly used at parsing time, later attributes can be set directly.

   report_invalid_syntax (message, level='ERROR')

   reset ()

   source
class robot.parsing.settings.Fixture
(setting_name, parent=None, comment=None)
Bases: robot.parsing.settings.Setting

is_set()

as_list()

directory

is_for_loop()

populate(value, comment=None)

Mainly used at parsing time, later attributes can be set directly.

report_invalid_syntax(message, level='ERROR')

reset()

source

class robot.parsing.settings.Timeout
(setting_name, parent=None, comment=None)
Bases: robot.parsing.settings.Setting

is_set()

as_list()

directory

is_for_loop()

populate(value, comment=None)

Mainly used at parsing time, later attributes can be set directly.

report_invalid_syntax(message, level='ERROR')

reset()

source

class robot.parsing.settings.Tags
(setting_name, parent=None, comment=None)
Bases: robot.parsing.settings.Setting

is_set()

as_list()

directory

is_for_loop()

populate(value, comment=None)

Mainly used at parsing time, later attributes can be set directly.

report_invalid_syntax(message, level='ERROR')

reset()

source

class robot.parsing.settings.Arguments
(setting_name, parent=None, comment=None)
Bases: robot.parsing.settings.Setting

as_list()

directory

is_for_loop()
`is_set()`

`populate(value, comment=None)`

Mainly used at parsing time, later attributes can be set directly.

`report_invalid_syntax(message, level='ERROR')`

`reset()`

`source`

class `robot.parsing.settings.Return(setting_name, parent=None, comment=None)`

Bases: `robot.parsing.settings.Setting`

`as_list()`

`directory`

`is_for_loop()`

`is_set()`

`populate(value, comment=None)`

Mainly used at parsing time, later attributes can be set directly.

`report_invalid_syntax(message, level='ERROR')`

`reset()`

`source`

class `robot.parsing.settings.Metadata(parent, name, value, comment=None, joined=False)`

Bases: `robot.parsing.settings.Setting`

`setting_name` = ‘Metadata’

`reset()`

`is_set()`

`as_list()`

`directory`

`is_for_loop()`

`populate(value, comment=None)`

Mainly used at parsing time, later attributes can be set directly.

`report_invalid_syntax(message, level='ERROR')`

`source`

class `robot.parsing.settings.Library(parent, name, args=None, alias=None, comment=None)`

Bases: `robot.parsing.settings._Import`

`as_list()`

`directory`

`is_for_loop()`

`is_set()`

`populate(value, comment=None)`

Mainly used at parsing time, later attributes can be set directly.

`report_invalid_syntax(message, level='ERROR')`
reset()
source
type
class robot.parsing.settings.Resource (parent, name, invalid_args=None, comment=None)
Bases: robot.parsing.settings._Import
as_list()
directory
is_for_loop()
is_set()
populate (value, comment=None)
    Mainly used at parsing time, later attributes can be set directly.
report_invalid_syntax (message, level='ERROR')
reset()
source
type
class robot.parsing.settings.Variables (parent, name, args=None, comment=None)
Bases: robot.parsing.settings._Import
as_list()
directory
is_for_loop()
is_set()
populate (value, comment=None)
    Mainly used at parsing time, later attributes can be set directly.
report_invalid_syntax (message, level='ERROR')
reset()
source
type
class robot.parsing.settings.ImportList (parent)
Bases: robot.parsing.settings._DataList
populate_library (data, comment)
populate_resource (data, comment)
populate_variables (data, comment)
add (meta)
class robot.parsing.settings.MetadataList (parent)
Bases: robot.parsing.settings._DataList
populate (name, value, comment)
add (meta)
4.10.9 tablepopulators Module

class robot.parsing.tablepopulators.Populator
    Bases: object
        Explicit interface for all populators.
        add(row)
        populate()

class robot.parsing.tablepopulators.NullPopulator
    Bases: robot.parsing.tablepopulators.Populator
        add(row)
        populate()

class robot.parsing.tablepopulators.SettingTablePopulator(table)
    Bases: robot.parsing.tablepopulators._TablePopulator
        add(row)
        populate()

class robot.parsing.tablepopulators.VariableTablePopulator(table)
    Bases: robot.parsing.tablepopulators._TablePopulator
        add(row)
        populate()

class robot.parsing.tablepopulators.TestTablePopulator(table)
    Bases: robot.parsing.tablepopulators._StepContainingTablePopulator
        add(row)
        populate()

class robot.parsing.tablepopulators.KeywordTablePopulator(table)
    Bases: robot.parsing.tablepopulators._StepContainingTablePopulator
        add(row)
        populate()

class robot.parsing.tablepopulators.ForLoopPopulator(for_loop_creator)
    Bases: robot.parsing.tablepopulators.Populator
        add(row)
        populate()

class robot.parsing.tablepopulators.TestCasePopulator(test_or_uk_creator)
    Bases: robot.parsing.tablepopulators._TestCaseUserKeywordPopulator
        add(row)
        populate()

class robot.parsing.tablepopulators.UserKeywordPopulator(test_or_uk_creator)
    Bases: robot.parsing.tablepopulators._TestCaseUserKeywordPopulator
        add(row)
        populate()
class robot.parsing.tablepopulators.VariablePopulator(setter, name)
    Bases: robot.parsing.tablepopulators._PropertyPopulator
    populate()
    add(row)

class robot.parsing.tablepopulators.SettingPopulator(setter)
    Bases: robot.parsing.tablepopulators._PropertyPopulator
    populate()
    add(row)

class robot.parsing.tablepopulators.DocumentationPopulator(setter)
    Bases: robot.parsing.tablepopulators._PropertyPopulator
    populate()
    add(row)

class robot.parsing.tablepopulators.MetadataPopulator(setter)
    Bases: robot.parsing.tablepopulators.DocumentationPopulator
    populate()
    add(row)

class robot.parsing.tablepopulators.StepPopulator(setter)
    Bases: robot.parsing.tablepopulators._PropertyPopulator
    populate()
    add(row)

4.10.10 tsvreader Module

class robot.parsing.tsvreader.TsvReader
    read(tsvfile, populator)
   classmethod split_row(row)

4.10.11 txtreader Module

class robot.parsing.txtreader.TxtReader
    Bases: robot.parsing.tsvreader.TsvReader
    classmethod split_row(row)
    read(tsvfile, populator)

4.11 reporting Package

4.11.1 reporting Package

Implements report and log file generation.
This package is considered stable.
4.11.2 jsbuildingcontext Module

```python
class robot.reporting.jsbuildingcontext.JsBuildingContext (log_path=None, split_log=False, prune_input=False):

    Bases: object

    string (string, escape=True)
    html (string)
    relative_source (source)
    timestamp (time)
    message_level (level)
    create_link_target (msg)
    link (msg)
    strings
    start_splitting_if_needed (split=False)
    end_splitting (model)
    prune_input (*args, **kwds)
```

4.11.3 jsexecutionresult Module

```python
class robot.reporting.jsexecutionresult.JsExecutionResult (suite, statistics, errors, strings, basemillis=None, split_results=None, min_level=None):

    Bases: object

    remove_data_not_needed_in_report ()
```

4.11.4 jsmodelbuilders Module

```python
class robot.reporting.jsmodelbuilders.JsModelBuilder (log_path=None, split_log=False, prune_input_to_save_memory=False):

    Bases: object

    build_from (result_from_xml)

class robot.reporting.jsmodelbuilders.SuiteBuilder (context):
    Bases: robot.reporting.jsmodelbuilders._Builder
    build (suite)

class robot.reporting.jsmodelbuilders.TestBuilder (context):
    Bases: robot.reporting.jsmodelbuilders._Builder
    build (test)

class robot.reporting.jsmodelbuilders.KeywordBuilder (context):
    Bases: robot.reporting.jsmodelbuilders._Builder
    build (kw, split=False)
```
class robot.reporting.jsmodelbuilders.MessageBuilder(context)
    Bases: robot.reporting.jsmodelbuilders._Builder
    build(msg)

class robot.reporting.jsmodelbuilders.StatisticsBuilder
    Bases: object
    build(statistics)

class robot.reporting.jsmodelbuilders.ErrorsBuilder(context)
    Bases: robot.reporting.jsmodelbuilders._Builder
    build(errors)

class robot.reporting.jsmodelbuilders.ErrorMessageBuilder(context)
    Bases: robot.reporting.jsmodelbuilders.MessageBuilder
    build(msg)

4.11.5 jswriter Module

class robot.reporting.jswriter.JsResultWriter(output, start_block='<script
type="text/javascript">n', end_block='</script>n', split_threshold=9500)
    Bases: object
    write(result, settings)

class robot.reporting.jswriter.SuiteWriter(write_json, split_threshold)
    Bases: object
    write(suite, variable)

class robot.reporting.jswriter.SplitLogWriter(output)
    Bases: object
    write(keywords, strings, index, notify)

4.11.6 logreportwriters Module

class robot.reporting.logreportwriters.LogWriter(js_model)
    Bases: robot.reporting.logreportwriters._LogReportWriter
    write(path, config)

class robot.reporting.logreportwriters.ReportWriter(js_model)
    Bases: robot.reporting.logreportwriters._LogReportWriter
    write(path, config)

class robot.reporting.logreportwriters.RobotModelWriter(output, model, config)
    Bases: robot.htmldata.htmlfilewriter.ModelWriter
    write(line)
    handles(line)
4.11.7 `outputwriter` Module

```python
class robot.reporting.outputwriter.OutputWriter(output):
    Bases: robot.output.xmllogger.XmlLogger, robot.result.visitor.ResultVisitor

    start_message(msg)
    close()
    start_errors(errors)
    end_errors(errors)
    end_result(result)
    start_total_statistics(total_stats)
    start_tag_statistics(tag_stats)
    start_suite_statistics(tag_stats)
    end_total_statistics(total_stats)
    end_tag_statistics(tag_stats)
    end_suite_statistics(tag_stats)
    visit_stat(stat)
    end_keyword(kw)
    end_message(msg)
    end_stat(stat)
    end_statistics(stats)
    end_suite(suite)
    end_suite_stats(tag_stats)
    end_tag_stats(tag_stats)
    end_test(test)
    end_total_stats(total_stats)
    log_message(msg)
    message(msg)
    set_log_level(level)
    start_keyword(kw)
    start_result(result)
    start_stat(stat)
    start_statistics(stats)
    start_suite(suite)
    start_suite_stats(tag_stats)
    start_tag_stats(tag_stats)
    start_test(test)
    start_total_stats(total_stats)
```
suite_stat (stat)
tag_stat (stat)
total_stat (stat)
visit_errors (errors)
visit_keyword (kw)
visit_message (msg)
visit_result (result)
visit_statistics (stats)
visit_suite (suite)
visit_suite_statistics (stats)
visit_tag_statistics (stats)
visit_test (test)
visit_total_statistics (stats)

4.11.8 resultwriter Module

class robot.reporting.resultwriter.ResultWriter (*data_sources)
   Bases: object
   write_results (settings, results=None)

class robot.reporting.resultwriter.Results (data_sources, settings)
   Bases: object
   result
   js_result

4.11.9 stringcache Module

class robot.reporting.stringcache.StringIndex
   Bases: long
   bit_length () \rightarrow \text{int or long}
       Number of bits necessary to represent self in binary.
       $\gg\gg\gg$ bin(37L) ‘0b100101’ $\gg\gg\gg$ (37L).bit_length() 6
   conjugate ()
       Returns self, the complex conjugate of any long.
   denominator
       the denominator of a rational number in lowest terms
   imag
       the imaginary part of a complex number
   numerator
       the numerator of a rational number in lowest terms
   real
       the real part of a complex number

4.11. reporting Package
class robot.reporting.stringcache.StringCache
    Bases: object
    add(text)
    dump()

4.11.10 xunitwriter Module

class robot.reporting.xunitwriter.XUnitWriter(execution_result)
    Bases: object
    write(output)

class robot.reporting.xunitwriter.XUnitFileWriter(xml_writer)
    Bases: robot.result.visitor.ResultVisitor
    Provides an xUnit-compatible result file.
    Attempts to adhere to the de facto schema guessed by Peter Reilly, see: http://marc.info/?l=ant-dev&m=123551933508682
    start_suite(suite)
    end_suite(suite)
    start_test(test)
    end_test(test)
    visit_keyword(kw)
    end_result(result)
    end_errors(errors)
    end_keyword(keyword)
    end_message(msg)
    end_stat(stat)
    end_statistics(stats)
    end_suite_statistics(suite_stats)
    end_tag_statistics(stats)
    end_total_statistics(stats)
    start_errors(errors)
    start_keyword(keyword)
    start_message(msg)
    start_result(result)
    start_stat(stat)
    start_statistics(stats)
    start_suite_statistics(stats)
    start_tag_statistics(stats)
    start_total_statistics(stats)
visit_errors (errors)
visit_message (msg)
visit_result (result)
visit_stat (stat)
visit_statistics (stats)
visit_suite (suite)
visit_suite_statistics (stats)
visit_tag_statistics (stats)
visit_test (test)
visit_total_statistics (stats)

class robot.reporting.xunitwriter.TestFailureWriter (writer)
    Bases: robot.result.visitor.ResultVisitor

    start_test (test)
    end_test (test)

    visit_message (msg)
        Populates the <failure> section, normally only with a ‘Stacktrace’.
        There is a weakness here because filtering is based on message level: - DEBUG level is used by RF for 'Tracebacks’ (what is expected here) - INFO and TRACE are used for keywords and arguments (not errors) - first FAIL message is already reported as <failure> attribute

    end_errors (errors)
    end_keyword (keyword)
    end_message (msg)
    end_result (result)
    end_stat (stat)
    end_statistics (stats)
    end_suite (suite)
    end_suite_statistics (suite_stats)
    end_tag_statistics (stats)
    end_total_statistics (stats)

start_errors (errors)
start_keyword (keyword)
start_message (msg)
start_result (result)
start_stat (stat)
start_statistics (stats)
start_suite (suite)
start_suite_statistics (stats)
4.12 result Package

4.12.1 result Package

Implements parsing results from XML output files. The entry point of this API is the `ExecutionResult()` factory method, which returns an instance of `Result`. This package is considered stable.

Example

The example below reads a given output file and marks each test case whose execution time is longer than three minutes failed. The `Result` object is then written back to disk and normal log and report files could be generated with `rebot` tool.

```python
#!/usr/bin/env python

"""Usage: check_test_times.py inpath [outpath]
Reads result of a test run from Robot output file and checks that no test took longer than 3 minutes to execute. If outpath is not given, the result is written over the original file.
"""

import sys
from robot.result import ExecutionResult

def check_tests(inpath, outpath=None):
    if not outpath:
        outpath = inpath
    result = ExecutionResult(inpath)
    _check_execution_times(result.suite)
    result.save(outpath)
```
def _check_execution_times(suite):
    for test in suite.tests:
        if test.status == 'PASS' and test.elapsedtime > 1000 * 60 * 3:
            test.status = 'FAIL'
            test.message = 'Test execution time was too long: {} seconds'.format(test.elapsedtime)
    for suite in suite.suites:
        _check_execution_times(suite)

if __name__ == '__main__':
    try:
        check_tests(*sys.argv[1:])
    except TypeError:
        print(__doc__)

4.12.2 configurer Module

class robot.result.configurer.SuiteConfigurer
    (name=None, doc=None, metadata=None, set_tags=None, include_tags=None, exclude_tags=None, include_suites=None, include_tests=None, process_empty_suite=False, remove_keywords=None, log_level=None, critical=None, noncritical=None, start_time=None, endtime=None)

    Bases: object

    add_tags
    remove_tags
    configure(suite)

4.12.3 executionerrors Module

class robot.result.executionerrors.ExecutionErrors
    (messages=None)

    Bases: object

    message_class
        alias of Message

    messages
    add(other)
    visit(visitor)

4.12.4 executionresult Module

class robot.result.executionresult.Result
    (source=None, root_suite=None, errors=None)

    Bases: object

    Contains results of test execution.

    Variables

        • source – Path to the xml file where results are read from.
• **suite** – Hierarchical `TestSuite` results.

• **errors** – Execution `ExecutionErrors`.

**statistics**
Test execution `Statistics`.

**return_code**
Return code (integer) of test execution.

`configure` *(status_rc=True, suite_config={}, stat_config={})*

`visit` *(visitor)*

`save` *(path=None)*

**class** `robot.result.executionresult.CombinedResult` *(others)*

Bases: `robot.result.executionresult.Result`

`add_result` *(other)*

`configure` *(status_rc=True, suite_config={}, stat_config={})*

**return_code**
Return code (integer) of test execution.

`save` *(path=None)*

**statistics**
Test execution `Statistics`.

`visit` *(visitor)*

### 4.12.5 keyword Module

**class** `robot.result.keyword.Keyword` *(name='', doc='', args=None, type='kw', timeout='', status='FAIL', starttime=None, endtime=None)*

Bases: `robot.model.keyword.Keyword`

Results of a single keyword.

**Variables**

• **name** – Keyword name.

• **parent** – `TestSuite` or `TestCase` that contains this keyword.

• **doc** – Keyword documentation.

• **args** – Keyword arguments, a list of strings.

• **type** – ‘SETUP’, ‘TEARDOWN’ or ‘KW’.

• **timeout** – Keyword timeout.

• **messages** – Log messages, a list of `Message` instances.

• **keywords** – Child keyword results, a list of `Keyword` instances

• **status** – String ‘PASS’ of ‘FAIL’.

• **starttime** – Keyword execution start time as a timestamp.

• **endtime** – Keyword execution end time as a timestamp.

**message_class**
alias of `Message`
status
starttime
endtime
message
elapsedtime
passed
FOR_ITEM_TYPE = ‘foritem’
FOR_LOOP_TYPE = ‘for’
KEYWORD_TYPE = ‘kw’
SETUP_TYPE = ‘setup’
TEARDOWN_TYPE = ‘teardown’
args
doc
id
keywords
messages
name
parent
timeout
type
visit (visitor)

4.12.6 keywordremover Module

robot.result.keywordremover.KeywordRemover (how)

class robot.result.keywordremover.AllKeywordsRemover
    Bases: robot.result.keywordremover._KeywordRemover
        visit_keyword (keyword)
        end_keyword (keyword)
        end_message (msg)
        end_suite (suite)
        end_test (test)
        start_keyword (keyword)
        start_message (msg)
        start_suite (suite)
        start_test (test)
        visit_message (msg)

4.12. result Package

413
class robot.result.keywordremover.PassedKeywordRemover
    Bases: robot.result.keywordremover._KeywordRemover

    visit_suite (suite)
    visit_test (test)

    class robot.result.keywordremover.ForLoopItemsRemover
        Bases: robot.result.keywordremover._KeywordRemover

        start_keyword (kw)
        visit_keyword (keyword)
        end_keyword (keyword)
        end_message (msg)
        end_suite (suite)
        end_test (test)

        start_message (msg)
        start_test (test)
        visit_keyword (kw)
        visit_message (msg)
        visit_suite (suite)
        visit_test (test)

    class robot.result.keywordremover.WaitUntilKeywordSucceedsRemover
        Bases: robot.result.keywordremover._KeywordRemover

        start_keyword (kw)
        visit_keyword (kw)
        end_keyword (keyword)
        end_message (msg)
        end_suite (suite)
        end_test (test)
start_message (msg)
start_suite (suite)
start_test (test)
visit_keyword (kw)
visit_message (msg)
visit_suite (suite)
visit_test (test)

class robot.result.keywordremover.ContainsWarning
    Bases: robot.model.visitor.SuiteVisitor
    start_suite (suite)
    start_test (test)
    start_keyword (keyword)
    visit_message (msg)
    end_keyword (keyword)
    end_message (msg)
    end_suite (suite)
    end_test (test)
    start_message (msg)
    visit_keyword (kw)
    visit_suite (suite)
    visit_test (test)

class robot.result.keywordremover.RemovalMessage (message)
    Bases: object
    set_if_removed (kw, len_before)
    set (kw, message=None)

4.12.7 message Module

class robot.result.message.Message (message='', level='INFO', html=False, timestamp=None, parent=None)
    Bases: robot.model.message.Message
    html
    html_message
    level
    message
    parent
    timestamp
    visit (visitor)
4.12.8 messagefilter Module

class robot.result.messagefilter.MessageFilter(loglevel)
   Bases: robot.model.visitor.SuiteVisitor

   start_keyword (keyword)
   end_keyword (keyword)
   end_message (msg)
   end_suite (suite)
   end_test (test)
   start_message (msg)
   start_suite (suite)
   start_test (test)
   visit_keyword (kw)
   visit_message (msg)
   visit_suite (suite)
   visit_test (test)

4.12.9 resultbuilder Module

robot.result.resultbuilder.ExecutionResult(*sources, **options)
   Constructs Result object based on execution result xml file(s).

   Parameters
   • sources – The Robot Framework output xml file(s).
   • options – Configuration options passed to ExecutionResultBuilder as keyword arguments. New in 2.7.5.

   Returns Result instance.

   See robot.result for usage example.

class robot.result.resultbuilder.ExecutionResultBuilder(source, in-
   clude_keywords=True)
   Bases: object

   build(result)

4.12.10 suiteteardownfailed Module

class robot.result.suiteteardownfailed.SuiteTeardownFailureHandler(suite_generator)
   Bases: robot.model.visitor.SuiteVisitor

   start_suite (suite)
   end_suite (suite)
   visit_test (test)
   visit_keyword (keyword)
end_keyword(keyword)
end_message(msg)
end_test(test)
start_keyword(keyword)
start_message(msg)
start_test(test)
visit_message(msg)
visit_suite(suite)

class robot.result.suitetearndownfailed.SuiteTeardownFailed(error)
Bases: robot.model.visitor.SuiteVisitor

visit_test(test)
visit_keyword(keyword)
end_keyword(keyword)
end_message(msg)
end_suite(suite)
end_test(test)
start_keyword(keyword)
start_message(msg)
start_suite(suite)
start_test(test)
visit_message(msg)
visit_suite(suite)

4.12.11 testcase Module

class robot.result.testcase.TestCase(name='', doc='', tags=None, timeout='', status='FAIL', message='', starttime=None, endtime=None)
Bases: robot.model.testcase.TestCase

Results of a single test case.

Variables

• name – Test case name.
• parent – TestSuite that contains this test.
• doc – Test case documentation.
• tags – Test case tags, a list of strings.
• timeout – Test case timeout.
• keywords – Keyword results, a list of Keyword instances and contains also possible setup and teardown keywords.
• status – String ‘PASS’ of ‘FAIL’.

4.12. result Package
• message – Possible failure message.
• starttime – Test case execution start time as a timestamp.
• endtime – Test case execution end time as a timestamp.

```python
keyword_class
    alias of Keyword

status
message
starttime
endtime
elapsedtime
passed
critical
doc
id
keywords
longname
name
parent
tags
timeout
visit(visitor)
```

### 4.12.12 testsuite Module

```python
class robot.result.testsuite.TestSuite(source='', name='', doc='', metadata=None, message='', starttime=None, endtime=None):
    Bases: robot.model.testsuite.TestSuite

Results of a single test suite.
```

**Variables**

- **parent** – Parent TestSuite or None.
- **source** – Path to the source file.
- **name** – Test suite name.
- **doc** – Test suite documentation.
- **metadata** – Test suite metadata as a dictionary.
- **suites** – Child suite results.
- **tests** – Test case results. a list of TestCase instances.
- **keywords** – A list containing setup and teardown results.
- **message** – Possible failure message.
• `starttime` – Test suite execution start time as a timestamp.
• `endtime` – Test suite execution end time as a timestamp.

test_class
  alias of `TestCase`

keyword_class
  alias of `Keyword`

message

starttime

endtime

status

statistics

full_message

elapsedtime

remove_keywords `(how)`

filter_messages `(log_level='TRACE')`

criticality

doc

filter `(included_suites=None, included_tests=None, included_tags=None, excluded_tags=None)`

id

keywords

longname

metadata

name

parent

set_criticality `(critical_tags=None, non_critical_tags=None)`

set_tags `(add=None, remove=None)`

source

suites

test_count

tests

visit `(visitor)`

4.12.13 visitor Module

class `robot.result.visitor.ResultVisitor`
  Bases: `robot.model.visitor.SuiteVisitor`

visit_result `(result)`

start_result `(result)`
end_result(result)
visit_statistics(stats)
start_statistics(stats)
end_statistics(stats)
visit_total_statistics(stats)
start_total_statistics(stats)
end_total_statistics(stats)
visit_tag_statistics(stats)
start_tag_statistics(stats)
end_tag_statistics(stats)
visit_suite_statistics(stats)
start_suite_statistics(stats)
end_suite_statistics(stats)
visit_stat(stat)
start_stat(stat)
end_stat(stat)
visit_errors(errors)
start_errors(errors)
end_errors(errors)
end_keyword(keyword)
end_message(msg)
end_suite(suite)
end_test(test)
start_keyword(keyword)
start_message(msg)
start_suite(suite)
start_test(test)
visit_keyword(kw)
visit_message(msg)
visit_suite(suite)
visit_test(test)

4.12.14 xmlelementhandlers Module

class robot.result.xmlelementhandlers(XmlElementHandler (execution_result,
root_handler=None)
Bases: object
start(elem)
end(elem)

class robot.result.xmlelementhandlers.RootHandler
    Bases: robot.result.xmlelementhandlers._Handler
    end(elem, result)
    handle_child(elem, result)
    start(elem, result)

class robot.result.xmlelementhandlers.RobotHandler
    Bases: robot.result.xmlelementhandlers._Handler
    tag = 'robot'
    start(elem, result)
    end(elem, result)
    handle_child(elem, result)

class robot.result.xmlelementhandlers_SUITEHandler
    Bases: robot.result.xmlelementhandlers._Handler
    tag = 'suite'
    start(elem, result)
    end(elem, result)
    handle_child(elem, result)

class robot.result.xmlelementhandlers.RootSuiteHandler
    Bases: robot.result.xmlelementhandlers.SuiteHandler
    start(elem, result)
    end(elem, result)
    handle_child(elem, result)
    tag = 'suite'

class robot.result.xmlelementhandlers_TestCaseHandler
    Bases: robot.result.xmlelementhandlers._Handler
    tag = 'test'
    start(elem, result)
    end(elem, result)
    handle_child(elem, result)

class robot.result.xmlelementhandlers.KeywordHandler
    Bases: robot.result.xmlelementhandlers._Handler
    tag = 'kw'
    start(elem, result)
    end(elem, result)
    handle_child(elem, result)

class robot.result.xmlelementhandlers.MessageHandler
    Bases: robot.result.xmlelementhandlers._Handler

4.12. result Package
tag = 'msg'
end(elem, result)
handle_child(elem, result)
start(elem, result)
class robot.result.xmlelementhandlers.KeywordStatusHandler
Bases: robot.result.xmlelementhandlers._StatusHandler
end(elem, result)
handle_child(elem, result)
start(elem, result)
tag = 'status'
class robot.result.xmlelementhandlers.SuiteStatusHandler
Bases: robot.result.xmlelementhandlers._StatusHandler
end(elem, result)
handle_child(elem, result)
start(elem, result)
tag = 'status'
class robot.result.xmlelementhandlers.TestStatusHandler
Bases: robot.result.xmlelementhandlers._StatusHandler
end(elem, result)
handle_child(elem, result)
start(elem, result)
tag = 'status'
class robot.result.xmlelementhandlers.DocHandler
Bases: robot.result.xmlelementhandlers._Handler
tag = 'doc'
end(elem, result)
handle_child(elem, result)
start(elem, result)
class robot.result.xmlelementhandlers.MetadataHandler
Bases: robot.result.xmlelementhandlers._Handler
tag = 'metadata'
end(elem, result)
handle_child(elem, result)
start(elem, result)
class robot.result.xmlelementhandlers.MetadataItemHandler
Bases: robot.result.xmlelementhandlers._Handler
tag = 'item'
end(elem, result)
handle_child(elem, result)
start(elem, result)
class robot.result.xmlElementHandlers.TagsHandler
Bases: robot.result.xmlElementHandlers._Handler
tag = 'tags'
end(elem, result)
handle_child(elem, result)
start(elem, result)
class robot.result.xmlElementHandlers.TagHandler
Bases: robot.result.xmlElementHandlers._Handler
tag = 'tag'
end(elem, result)
handle_child(elem, result)
start(elem, result)
class robot.result.xmlElementHandlers.ArgumentsHandler
Bases: robot.result.xmlElementHandlers._Handler
tag = 'arguments'
end(elem, result)
handle_child(elem, result)
start(elem, result)
class robot.result.xmlElementHandlers.ArgumentHandler
Bases: robot.result.xmlElementHandlers._Handler
tag = 'arg'
end(elem, result)
handle_child(elem, result)
start(elem, result)
class robot.result.xmlElementHandlers.ErrorsHandler
Bases: robot.result.xmlElementHandlers._Handler
tag = 'errors'
start(elem, result)
end(elem, result)
handle_child(elem, result)
class robot.result.xmlElementHandlers.StatisticsHandler
Bases: robot.result.xmlElementHandlers._Handler
tag = 'statistics'
handle_child(elem, result)
end(elem, result)
start(elem, result)
4.13 running Package

4.13.1 running Package

Implements the core test execution logic.

The code in this package is in many places suboptimal and likely to change in RF 2.8. External code should use this package with care.

Currently, the main entry point is the TestSuite() factory method.

robot.running.UserLibrary(path)

Create a user library instance from given resource file.

This is used at least by libdoc.py.

4.13.2 arguments Module

class robot.running.arguments.PythonKeywordArguments (argument_source, kw_or_lib_name)

Bases: robot.running.arguments._KeywordArguments

check_arg_limits (args, namedargs={})

check_arg_limits_for_dry_run (args)

resolve (args, variables, output=None)

class robot.running.arguments.JavaKeywordArguments (handler_method, name)

Bases: robot.running.arguments._KeywordArguments

check_arg_limits (args, namedargs={})

check_arg_limits_for_dry_run (args)

resolve (args, variables, output=None)

class robot.running.arguments.DynamicKeywordArguments (argument_source, kw_or_lib_name)

Bases: robot.running.arguments._KeywordArguments

check_arg_limits (args, namedargs={})

check_arg_limits_for_dry_run (args)

resolve (args, variables, output=None)

class robot.running.arguments.RunKeywordArguments (argument_source, name, arg_resolution_index)

Bases: robot.running.arguments.PythonKeywordArguments

check_arg_limits (args, namedargs={})

check_arg_limits_for_dry_run (args)

resolve (args, variables, output=None)

class robot.running.arguments.PythonInitArguments (argument_source, kw_or_lib_name)

Bases: robot.running.arguments.PythonKeywordArguments

check_arg_limits (args, namedargs={})

check_arg_limits_for_dry_run (args)
resolve(args, variables, output=None)

class robot.running.arguments.JavaInitArguments(handler_method, name)
    Bases: robot.running.arguments.JavaKeywordArguments
    resolve(args, variables=None)
    check_arg_limits(args, namedargs={})
    check_arg_limits_for_dry_run(args)

class robot.running.arguments.UserKeywordArguments(args, name)
    Bases: object
    resolve_arguments_for_dry_run(arguments)
    resolve(arguments, variables, output)
    set_variables(arg_values, variables, output)

class robot.running.arguments.UserKeywordArgsTemplate(minargs, defaults)
    Bases: object
    set_value(idx, value)
    as_list()

class robot.running.arguments.UserKeywordArgumentResolver(arguments)
    Bases: robot.running.arguments._ArgumentResolver
    resolve(values, output, variables=None)

class robot.running.arguments.JavaKeywordArgumentResolver(arguments)
    Bases: object
    resolve(values, output, variables=None)

class robot.running.arguments.PythonKeywordArgumentResolver(arguments)
    Bases: robot.running.arguments._ArgumentResolver
    resolve(values, output, variables=None)

class robot.running.arguments.JavaKeywordArgumentResolver(arguments)
    Bases: object
    resolve(values, output, variables)

4.13.3 context Module

class robot.running.context.ExecutionContexts
    Bases: object
    current
    top
    namespaces
    start_suite(namespace, output, dry_run=False)
    end_suite()

4.13.4 defaultvalues Module

class robot.running.defaultvalues.DefaultValues(settings, parent_default_values=None)
    Bases: object
    get_setup(tc_setup)
get_teardown (tc_teardown)
get_timeout (tc_timeout)
get_tags (tc_tags)
get_template (template)

4.13.5 fixture Module

class robot.running.fixture.Setup (name, args)
    Bases: robot.running.fixture._Fixture
    replace_variables (variables, errors)
    run (context)
    serialize (serializer)

class robot.running.fixture.Teardown (name, args)
    Bases: robot.running.fixture._Fixture
    replace_variables (variables, errors)
    run (context)
    serialize (serializer)

4.13.6 handlers Module

robot.running.handlers.Handler (library, name, method)
robot.running.handlers.DynamicHandler (library, name, method, doc, argspec)
robot.running.handlers.InitHandler (library, method, docgetter=None)

4.13.7 importer Module

class robot.running.importer.Importer
    Bases: object
    reset ()
    import_library (name, args=None, alias=None, variables=None)
    import_resource (path)

class robot.running.importer.ImportCache
    Keeps track on and optionally caches imported items.
    Handles paths in keys case-insensitively on case-insensitive OSes. Unlike dicts, this storage accepts mutable values in keys.
    add (key, item=None)
    values ()
4.13.8 javaargcoercer Module

4.13.9 keywords Module

class robot.running.keywords.Keywords(steps, template=None)
    Bases: object
    add_keyword(keyword)
    run(context)

class robot.running.keywords.Keyword(name, args, assign=None, type='kw')
    Bases: robot.common.keyword.BaseKeyword
    run(context)
    passed
    serialize(serializer)

class robot.running.keywords.ForLoop(forstep, template=None)
    Bases: robot.common.keyword.BaseKeyword
    run(context)
    passed
    serialize(serializer)

4.13.10 model Module

robot.running.model.TestSuite(datasources, settings, process_variables=True)
    Creates a runnable test suite from given data sources and settings.

    This is a factory method that returns either RunnableTestSuite or RunnableMultiTestSuite depending on one or more data sources given. This method, and especially the returned suite, is likely to change heavily in version 2.8.

    Parameters
    • datasources – List of paths to read data from. Starting from 2.7.2, a single datasource can also be given as a string.
    • settings (RobotSettings) – Execution configuration.

    Returns RunnableTestSuite

class robot.running.model.RunnableTestSuite(data, parent=None, defaults=None, process_variables=True)
    Bases: robot.common.model.BaseTestSuite
    filter_emptySuites()
    run(output, parent_context=None, errors=None)
    filter(suites=None, tests=None, includes=None, excludes=None, zero_tests_ok=False)
    filter_by_names(suites=None, tests=None, zero_tests_ok=False)
    filter_by_tags(includes=None, excludes=None, zero_tests_ok=False)
    get_full_message()
        Returns suite’s message including statistics message
    get_stat_message()
get_test_count()

id
longname
name

return_code

serialize(serializer)

set_critical_tags(critical, non_critical)

set_doc(doc)

set_metadata(metalist)

set_name(name)

set_options(settings)

set_runmode(runmode)

set_status()

Sets status and statistics based on subsuite and test statuses.
Can/should be used when statuses have been changed somehow.

set_tags(tags)

suite_teardown_failed(error=None, message=None)

class robot.running.modelRunnableMultiTestSuite(suitedatas, process_variables=True)
    Bases: robot.running.model.RunnableTestSuite

filter(suites=None, tests=None, includes=None, excludes=None, zero_tests_ok=False)

filter_by_names(suites=None, tests=None, zero_tests_ok=False)

filter_by_tags(includes=None, excludes=None, zero_tests_ok=False)

filter_emptySuites()

get_full_message()

Returns suite’s message including statistics message

gstat_message()

gtest_count()

id
longname
name

return_code

run(output, parent_context=None, errors=None)

serialize(serializer)

set_critical_tags(critical, non_critical)

set_doc(doc)

set_metadata(metalist)

set_name(name)
set_options (settings)
set_runmode (runmode)
set_status ()
    Sets status and statistics based on subsuite and test statuses.
    Can/should be used when statuses have been changed somehow.
set_tags (tags)
suite_teardown_failed (error=None, message=None)

class robot.running.model.RunnableTestCase (tc_data, parent, defaults)
    Bases: robot.common.model.BaseTestCase
    run (context, parent_errors)
    id
    is_included (incl_tags, excl_tags)
        Returns True if this test case is included but not excluded.
        If no ‘incl_tags’ are given all tests are considered to be included.
    longname
    passed
    serialize (serializer)
    set_criticality (critical)
    suite_teardown_failed (message)

4.13.11 namespace Module

class robot.running.namespace.Namespace (suite, parent_vars)
    handle_imports ()
    import_resource (name, overwrite=True)
    import_variables (name, args, overwrite=False, variables=None)
    import_library (name, args=None, alias=None, variables=None)
    start_test (test)
    end_test ()
    set_test_status_before_teardown (message, status)
    end_suite ()
    start_user_keyword (handler)
    end_user_keyword ()
    get_library_instance (libname)
    get_handler (name)
4.13.12 outputcapture Module

class robot.running.outputcapture.OutputCapturer(library_import=False)
    release_and_log()

4.13.13 runerrors Module

class robot.running.runerrors.SuiteRunErrors(exit_on_failure_mode=False, skip_teardowns_on_exit_mode=False)
    Bases: object
    exit
    start_suite()
    end_suite()
    is_setup_allowed()
    is_teardown_allowed()
    suite_initialized(error=None)
    setup_executed(error=None)
    get_suite_error()
    get_child_error()
    test_failed(exit=False, critical=False)

class robot.running.runerrors.TestRunErrors(parent)
    Bases: object
    is_run_allowed()
    is_teardown_allowed()
    test_initialized(error=None)
    setup_failed(error)
    keyword_failed(error)
    teardown_failed(error)
    test_failed(exit=False, critical=False)
    get_message()
    get_teardown_message(message)
    get_parent_or_init_error()
4.13.14 runkwregister Module

4.13.15 signalhandler Module

4.13.16 testlibraries Module

`robot.running.testlibraries.TestLibrary(name, args=None, variables=None, create_handlers=True)`

4.13.17 userkeyword Module

```python
class robot.running.userkeyword.UserLibrary (user_keywords, path=None)
    Bases: robot.common.libraries.BaseLibrary
    supports_named_arguments = True
    has_handler (name)
    get_handler (name)

class robot.running.userkeyword.UserKeywordHandler (keyword, libname)
    Bases: object
    type = 'user'
    longname
    shortdoc
    init_keyword (varz)
    run (context, arguments)

class robot.running.userkeyword.EmbeddedArgsTemplate (keyword, libname)
    Bases: robot.running.userkeyword.UserKeywordHandler
    init_keyword (varz)
    longname
    run (context, arguments)
    shortdoc
    type = 'user'

class robot.running.userkeyword.EmbeddedArgs (name, template)
    Bases: robot.running.userkeyword.UserKeywordHandler
    init_keyword (varz)
    longname
    run (context, arguments)
    shortdoc
    type = 'user'
```

4.13. running Package
4.13.18 Subpackages

timeouts Package

timeouts Package

class robot.running.timeouts.TestTimeout (timeout=None, message='', variables=None)
   Bases: robot.running.timeouts._Timeout
   type = 'Test'
   set_keyword_timeout (timeout_occurred)
   any_timeout_occurred ()
   active
   get_message ()
   replace_variables (variables)
   run (runnable, args=None, kwargs=None)
   start ()
   time_left ()
   timed_out ()

class robot.running.timeouts.KeywordTimeout (timeout=None, message='', variables=None)
   Bases: robot.running.timeouts._Timeout
   type = 'Keyword'
   active
   get_message ()
   replace_variables (variables)
   run (runnable, args=None, kwargs=None)
   start ()
   time_left ()
   timed_out ()

stoppablethread Module

class robot.running.timeouts.stoppablethread.Thread (runner, name=None)
   Bases: threading.Thread
   A subclass of threading.Thread, with a stop() method.
   Original version posted by Connelly Barnes to python-list and available at http://mail.python.org/pipermail/python-list/2004-May/219465.html
   This version mainly has kill() changed to stop() to match java.lang.Thread.
   This is a hack but seems to be the best way the get this done. Only used in Python because in Jython we can use java.lang.Thread.
   start ()
stop()

daemon
A boolean value indicating whether this thread is a daemon thread (True) or not (False).
This must be set before start() is called, otherwise RuntimeError is raised. Its initial value is inherited from
the creating thread; the main thread is not a daemon thread and therefore all threads created in the main
thread default to daemon = False.
The entire Python program exits when no alive non-daemon threads are left.

getName()

ident
Thread identifier of this thread or None if it has not been started.
This is a nonzero integer. See the thread.get_ident() function. Thread identifiers may be recycled when a
thread exits and another thread is created. The identifier is available even after the thread has exited.

isAlive()
Return whether the thread is alive.
This method returns True just before the run() method starts until just after the run() method terminates.
The module function enumerate() returns a list of all alive threads.

isDaemon()
is_alive()
Return whether the thread is alive.
This method returns True just before the run() method starts until just after the run() method terminates.
The module function enumerate() returns a list of all alive threads.

join(timeout=None)
Wait until the thread terminates.
This blocks the calling thread until the thread whose join() method is called terminates – either normally
or through an unhandled exception or until the optional timeout occurs.
When the timeout argument is present and not None, it should be a floating point number specifying a
timeout for the operation in seconds (or fractions thereof). As join() always returns None, you must call
isAlive() after join() to decide whether a timeout happened – if the thread is still alive, the join() call timed
out.
When the timeout argument is not present or None, the operation will block until the thread terminates.
A thread can be join()ed many times.
join() raises a RuntimeError if an attempt is made to join the current thread as that would cause a deadlock.
It is also an error to join() a thread before it has been started and attempts to do so raises the same exception.

name
A string used for identification purposes only.
It has no semantics. Multiple threads may be given the same name. The initial name is set by the construc-
tor.

run()
Method representing the thread’s activity.
You may override this method in a subclass. The standard run() method invokes the callable object passed
to the object’s constructor as the target argument, if any, with sequential and keyword arguments taken
from the args and kwargs arguments, respectively.

setDaemon(daemonic)
setName(name)

timeoutsigaling Module

class robot.running.timeouts.timeoutsignaling.Timeout(timeout, error)
    Bases: object
    execute(runnable)

timeoutthread Module

class robot.running.timeouts.timeoutthread.ThreadedRunner(runnable)
    Bases: object
    run()
    run_in_thread(timeout)
    get_result()
    stop_thread()

class robot.running.timeouts.timeoutthread.Timeout(timeout, error)
    Bases: object
    execute(runnable)

timeoutwin Module

class robot.running.timeouts.timeoutwin.Timeout(timeout, timeout_error)
    Bases: object
    execute(runnable)

4.14 utils Package

4.14.1 utils Package

Various generic utility classes and functions.
Provided utilities are generally stable, but absolute backwards compatibility between major versions is not guaranteed.

4.14.2 application Module

class robot.utils.application.Application(usage, name=None, version=None, arg_limits=None, logger=None, **auto_options)
execute(*arguments, **options)

class robot.utils.application.DefaultLogger
    Bases: object

    register_file_logger()
    info(message)
    error(message)
    close()

4.14.3 argumentparser Module

class robot.utils.argumentparser.ArgumentParser(usage, name=None, version=None, arg_limits=None, validator=None, auto_help=True, auto_version=True, auto_escape=True, auto_pythonpath=True, auto_argumentfile=True)

Available options and tool name are read from the usage.

Tool name is got from the first row of the usage. It is either the whole row or anything before first ‘–’.

parse_args(args_list)
    Parse given arguments and return options and positional arguments.

    Arguments must be given as a list and are typically sys.argv[1:]

    Options are returned as a dictionary where long options are keys. Value is a string for those options that can be given only one time (if they are given multiple times the last value is used) or None if the option is not used at all. Value for options that can be given multiple times (denoted with ‘*’ in the usage) is a list which contains all the given values and is empty if options are not used. Options not taken arguments have value False when they are not set and True otherwise.

    Positional arguments are returned as a list in the order they are given.

    If ‘check_args’ is True, this method will automatically check that correct number of arguments, as parsed from the usage line, are given. If the last argument in the usage line ends with the character ‘s’, the maximum number of arguments is infinite.

    Possible errors in processing arguments are reported using DataError.

    Some options have a special meaning and are handled automatically if defined in the usage and given from the command line:

    –escape option can be used to automatically unescape problematic characters given in an escaped format.

    –argumentfile can be used to automatically read arguments from a specified file. When –argumentfile is used, the parser always allows using it multiple times. Adding ‘*’ to denote that is thus recommend. A special value ‘stdin’ can be used to read arguments from stdin instead of a file.

    –pythonpath can be used to add extra path(s) to sys.path.

    –help and –version automatically generate help and version messages. Version is generated based on the tool name and version – see __init__ for information how to set them. Help contains the whole usage given to __init__. Possible <VERSION> text in the usage is replaced with the given version. Possible <—ESCAPES—> is replaced with available escapes so that they are wrapped to multiple lines but take the same amount of horizontal space as <—ESCAPES—>. Both help and version are wrapped to Information exception.
class robot.utils.argumentparser.ArgLimitValidator(arg_limits)
    Bases: object

class robot.utils.argumentparser.ArgFileParser(options)
    Bases: object
        process(args)

4.14.4 asserts Module

Convenience functions for testing both in unit and higher levels.

Benefits:
- Integrates 100% with unittest (see example below)
- Can be easily used without unittest (using unittest.TestCase when you only need convenient asserts is not so nice)
- Saved typing and shorter lines because no need to have ‘self.’ before asserts. These are static functions after all so that is OK.
- All ‘equals’ methods (by default) report given values even if optional message given. This behavior can be controlled with the optional values argument.

Drawbacks:
- unittest is not able to filter as much non-interesting traceback away as with its own methods because AssertionError occur outside

Most of the functions are copied more or less directly from unittest.TestCase which comes with the following license.
Further information about unittest in general can be found from http://pyunit.sourceforge.net/. This module can be used freely in same terms as unittest.

unittest license:

Copyright (c) 1999-2003 Steve Purcell
This module is free software, and you may redistribute it and/or modify it under the same terms as Python itself, so long as this copyright message and disclaimer are retained in their original form.

IN NO EVENT SHALL THE AUTHOR BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF THIS CODE, EVEN IF THE AUTHOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

THE AUTHOR SPECIFICALLY DISCLAIMS ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE CODE PROVIDED HEREUNDER IS ON AN "AS IS" BASIS, AND THERE IS NO OBLIGATION WHATSOEVER TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

Examples:

import unittest
from robot.util.asserts import *

class MyTests(unittest.TestCase):
    def test_old_style(self):
        self.assertEquals(1, 2, 'my msg')
```python
def test_new_style(self):
    assert_equals(1, 2, 'my msg')
```

Example output:

```
FF
======================================================================
FAIL: test_old_style (__main__.MyTests)
Traceback (most recent call last):
  File "example.py", line 7, in test_old_style
    self.assertEquals(1, 2, 'my msg')
AssertionError: my msg
```

```
FAIL: test_new_style (__main__.MyTests)
Traceback (most recent call last):
  File "example.py", line 10, in test_new_style
    assert_equals(1, 2, 'my msg')
File "/path/to/robot/asserts.py", line 142, in fail_unless_equal
  _report_unequality_failure(first, second, msg, values, '!=')
File "/path/to/robot/src/robot/asserts.py", line 209, in _report_unequality_failure
  raise _report_failure(msg)
File "/path/to/robot/src/robot/asserts.py", line 200, in _report_failure
  raise AssertionError(msg)
AssertionError: my msg: 1 != 2
```

Ran 2 tests in 0.000s

FAILED (failures=2)

```python
robot.utils.asserts.fail(msg=None)
Fail test immediately with the given message.

robot.utils.asserts.error(msg=None)
Error test immediately with the given message.

robot.utils.asserts.fail_if(expr, msg=None)
Fail the test if the expression is True.

robot.utils.asserts.fail_unless(expr, msg=None)
Fail the test unless the expression is True.

robot.utils.asserts.fail_if_none(obj, msg=None, values=True)
Fail the test if given object is None.

robot.utils.asserts.fail_unless_none(obj, msg=None, values=True)
Fail the test if given object is not None.

robot.utils.asserts.fail_unless_raises(exc_class, callable_obj, *args, **kwargs)
Fail unless an exception of class exc_class is thrown by callable_obj.

callable_obj is invoked with arguments args and keyword arguments kwargs. If a different type of exception is thrown, it will not be caught, and the test case will be deemed to have suffered an error, exactly as for an unexpected exception.

If a correct exception is raised, the exception instance is returned by this method.
```
robot.utils.asserts.fail_unless_raises_with_msg(exc_class, expected_msg, callable_obj, *args, **kwargs)

Similar to fail_unless_raises but also checks the exception message.

robot.utils.asserts.fail_unless_equal(first, second, msg=None, values=True)

Fail if given objects are unequal as determined by the ‘==’ operator.

robot.utils.asserts.fail_if_equal(first, second, msg=None, values=True)

Fail if given objects are equal as determined by the ‘==’ operator.

robot.utils.asserts.fail_unless_almost_equal(first, second, places=7, msg=None, values=True)

Fail if the two objects are unequal after rounded to given places. Unequality is determined by object’s difference rounded to the given number of decimal places (default 7) and comparing to zero. Note that decimal places (from zero) are usually not the same as significant digits (measured from the most significant digit).

robot.utils.asserts.fail_if_almost_equal(first, second, places=7, msg=None, values=True)

Fail if the two objects are unequal after rounded to given places. Equality is determined by object’s difference rounded to to the given number of decimal places (default 7) and comparing to zero. Note that decimal places (from zero) are usually not the same as significant digits (measured from the most significant digit).

robot.utils.asserts.assert_equal(first, second, msg=None, values=True)

Fail if given objects are unequal as determined by the ‘==’ operator.

robot.utils.asserts.assert_equals(first, second, msg=None, values=True)

Fail if given objects are unequal as determined by the ‘==’ operator.

robot.utils.asserts.assert_not_equal(first, second, msg=None, values=True)

Fail if given objects are equal as determined by the ‘==’ operator.

robot.utils.asserts.assert_not_equals(first, second, msg=None, values=True)

Fail if given objects are equal as determined by the ‘==’ operator.

robot.utils.asserts.assert_almost_equal(first, second, places=7, msg=None, values=True)

Fail if the two objects are unequal after rounded to given places. Unequality is determined by object’s difference rounded to the given number of decimal places (default 7) and comparing to zero. Note that decimal places (from zero) are usually not the same as significant digits (measured from the most significant digit).

robot.utils.asserts.assert_almost_equals(first, second, places=7, msg=None, values=True)

Fail if the two objects are unequal after rounded to given places. Unequality is determined by object’s difference rounded to the given number of decimal places (default 7) and comparing to zero. Note that decimal places (from zero) are usually not the same as significant digits (measured from the most significant digit).

robot.utils.asserts.assert_not_almost_equal(first, second, places=7, msg=None, values=True)

Fail if the two objects are unequal after rounded to given places. Equality is determined by object’s difference rounded to to the given number of decimal places (default 7) and comparing to zero. Note that decimal places (from zero) are usually not the same as significant digits (measured from the most significant digit).

robot.utils.asserts.assert_not_almost_equals(first, second, places=7, msg=None, values=True)

Fail if the two objects are unequal after rounded to given places.
Equality is determined by object’s difference rounded to to the given number of decimal places (default 7) and comparing to zero. Note that decimal places (from zero) are usually not the same as significant digits (measured from the most significant digit).

```python
robot.utils.asserts.assert_raises(exc_class, callable_obj, *args, **kwargs)
```

Fail unless an exception of class exc_class is thrown by callable_obj.

callable_obj is invoked with arguments args and keyword arguments kwargs. If a different type of exception is thrown, it will not be caught, and the test case will be deemed to have suffered an error, exactly as for an unexpected exception.

If a correct exception is raised, the exception instance is returned by this method.

```python
robot.utils.asserts.assert_raises_with_msg(exc_class, expected_msg, callable_obj, *args, **kwargs)
```

Similar to fail_unless_raises but also checks the exception message.

```python
robot.utils.asserts.assert_(expr, msg=None)
```

Fail the test unless the expression is True.

```python
robot.utils.asserts.assert_true(expr, msg=None)
```

Fail the test unless the expression is True.

```python
robot.utils.asserts.assert_false(expr, msg=None)
```

Fail the test if the expression is True.

```python
robot.utils.asserts.assert_none(obj, msg=None, values=True)
```

Fail the test if given object is not None.

```python
robot.utils.asserts.assert_not_none(obj, msg=None, values=True)
```

Fail the test if given object is None.

### 4.14.5 charwidth Module

A module to handle different character widths on the console.

Some East Asian characters have width of two on console, and combining characters themselves take no extra space.

See issue 604 [1] for more details about East Asian characters. The issue also contains `generate_wild_chars.py` script that was originally used to create `_EAST_ASIAN_WILD_CHARS` mapping. An updated version of the script is attached to issue 1096. Big thanks for xiéyánbó for the script and the original patch.

Note that Python’s `unicodedata` module is not used here because importing it takes several seconds on Jython.


```python
robot.utils.charwidth.get_char_width(char)
```

### 4.14.6 compress Module

```python
robot.utils.compress.compress_text(text)
```

### 4.14.7 connectioncache Module

```python
class robot.utils.connectioncache.ConnectionCache(no_current_msg='No open connection')
```

Connection cache for different Robot test libraries that use connections.

4.14. utils Package 439
This cache stores connections and allows switching between them using generated indexes or user given aliases. Can be used for example by web testing libraries where there’s need for multiple concurrent connections.

Note that in most cases there should be only one instance of this class but this is not enforced.

register (connection, alias=None)

Registers given connection with optional alias and returns its index.

Given connection is set to be the current connection. Alias must be a string. The index of the first connection after initialization or close_all or empty_cache is 1, second is 2, etc.

switch (index_or_alias)

Switches to the connection specified by given index or alias.

If alias is given it must be a string. Indexes can be either integers or strings that can be converted into integer. Raises RuntimeError if no connection with given index or alias found.

close_all (closer_method='close')

Closes connections using given closer method and empties cache.

If simply calling the closer method is not adequate for closing connections, clients should close connections themselves and use empty_cache afterwards.

empty_cache()

Empties the connections cache.

Indexes of new connections starts from 1 after this.

4.14.8 encoding Module

robot.utils.encoding.decode_output (string)

Decodes bytes from console encoding to Unicode.

robot.utils.encoding.encode_output (string, errors='replace')

Encodes Unicode to bytes in console encoding.

robot.utils.encoding.decode_from_system (string, can_be_from_java=True)

Decodes bytes from system (e.g. cli args or env vars) to Unicode.

robot.utils.encoding.encode_to_system (string, errors='replace')

Encodes Unicode to system encoding (e.g. cli args and env vars).

robot.utils.encoding.utf8open (*args, **kwds)

4.14.9 encodingsniffer Module

robot.utils.encodingsniffer.get_system_encoding()

robot.utils.encodingsniffer.get_output_encoding()

4.14.10 error Module

robot.utils.error.get_error_message()

Returns error message of the last occurred exception.

This method handles also exceptions containing unicode messages. Thus it MUST be used to get messages from all exceptions originating outside the framework.
robot.utils.error.get_error_details()
    Returns error message and details of the last occurred exception.

robot.utils.error.ErrorDetails()
    This factory returns an object that wraps the last occurred exception
    It has attributes message, traceback and error, where message contains type and message of the original error,
    traceback contains the traceback/stack trace and error contains the original error instance.

class robot.utils.error.PythonErrorDetails(exc_type, exc_value, exc_traceback)
    Bases: robot.utils.error._ErrorDetails
    message
    traceback

class robot.utils.error.JavaErrorDetails(exc_type, exc_value, exc_traceback)
    Bases: robot.utils.error._ErrorDetails
    message
    traceback

4.14.11 escaping Module

robot.utils.escaping.escape(item)
robot.utils.escaping.unescape(item)

4.14.12 etreewrapper Module

class robot.utils.etreewrapper.ETSource(source)
    Bases: object

4.14.13 htmlformatters Module

class robot.utils.htmlformatters.LinkFormatter
    Bases: object
    format_url(text)
    format_link(text)

class robot.utils.htmlformatters.LineFormatter
    Bases: object
    handles(line)
    newline = "\n"
    format(line)

class robot.utils.htmlformatters.HtmlFormatter
    Bases: object
    format(text)

class robot.utils.htmlformatters.RulerFormatter
    Bases: robot.utils.htmlformatters._SingleLineFormatter
    format_line(line)
add (line)
end()
format (lines)
handles (line)
class robot.utils.htmlformatters.HeaderFormatter
Bases: robot.utils.htmlformatters._SingleLineFormatter
format_line (line)
add (line)
end()
format (lines)
handles (line)
class robot.utils.htmlformatters.ParagraphFormatter (other_formatters)
Bases: robot.utils.htmlformatters._Formatter
format (lines)
add (line)
end()
handles (line)
class robot.utils.htmlformatters.TableFormatter
Bases: robot.utils.htmlformatters._Formatter
format (lines)
add (line)
end()
handles (line)
class robot.utils.htmlformatters.PreformattedFormatter
Bases: robot.utils.htmlformatters._Formatter
format (lines)
add (line)
end()
handles (line)
class robot.utils.htmlformatters.ListFormatter
Bases: robot.utils.htmlformatters._Formatter
format (lines)
add (line)
end()
handles (line)
4.14.14 importer Module

class robot.utils.importer.Importer(type=None, logger=None)
    Bases: object

    import_class_or_module(name, instantiate_with_args=None)
    Imports Python class/module or Java class with given name.

    Class can either live in a module/package or be standalone Java class. In the former case the name is
    something like ‘MyClass’ and in the latter it could be ‘your.package.YourLibrary’. Python classes always
    live in a module, but if the module name is exactly same as the class name then simple ‘MyLibrary’ will
    import a class.

    Python modules can be imported both using format ‘MyModule’ and ‘mymodule.submodule’.

    name can also be a path to the imported file/directory. In that case importing is done using import_class_or_module_by_path
    method.

    If instantiate_with_args is not None, imported classes are instantiated with the specified arguments auto-
    matically.

    import_class_or_module_by_path(path, instantiate_with_args=None)
    Import a Python module or Java class using a file system path.

    When importing a Python file, the path must end with ‘.py’ and the actual file must also exist. When
    importing a Python module implemented as a directory, the path must end with ‘/’ or, on Windows, with
    ‘\’.

    When importing Java classes, the path must end with ‘.java’ or ‘.class’. The class file must exist in both
    cases and in the former case also the source file must exist.

    If instantiate_with_args is not None, imported classes are instantiated with the specified arguments auto-
    matically.

class robot.utils.importer.ByPathImporter(logger)
    Bases: robot.utils.importer._Importer

    handles(path)

    import_(path)

class robot.utils.importer.NonDottedImporter(logger)
    Bases: robot.utils.importer._Importer

    handles(name)

    import_(name)

class robot.utils.importer.DottedImporter(logger)
    Bases: robot.utils.importer._Importer

    handles(name)

    import_(name)

4.14.15 markuputils Module

    robot.utils.markuputils.html_escape(text)
    robot.utils.markuputils.xml_escape(text)
    robot.utils.markuputils.html_format(text)
robot.utils.markuputils.attribute_escape(attr)

### 4.14.16 markupwriters Module

class robot.utils.markupwriters.HtmlWriter(output, line_separator='\n', encoding=None)
   Bases: robot.utils.markupwriters._MarkupWriter

   Parameters
   
   • output – Either an opened, file like object, or a path to the desired output file. In the latter case, the file is created and clients should use close() method to close it.
   
   • line_separator – Defines the used line separator.
   
   • encoding – Encoding to be used to encode all text written to the output file. If None, text will not be encoded.

   close()
      Closes the underlying output file.

   content(content=None, escape=True)

   element(name, content=None, attrs=None, escape=True, newline=True)

   end(name, newline=True)

class robot.utils.markupwriters.XmlWriter(output, line_separator='\n', encoding=None)
   Bases: robot.utils.markupwriters._MarkupWriter

   close()
      Closes the underlying output file.

   content(content=None, escape=True)

   element(name, content=None, attrs=None, escape=True, newline=True)

   end(name, newline=True)

   start(name, attrs=None, newline=True)

class robot.utils.markupwriters.NullMarkupWriter(*args)
   Bases: object

   Null implementation of _MarkupWriter interface

   start(*args)

   content(*args)

   element(*args)

   end(*args)

   close(*args)

### 4.14.17 match Module

robot.utils.match.eq(str1, str2, ignore=(), caseless=True, spaceless=True)

robot.utils.match.matches(string, pattern, ignore=(), caseless=True, spaceless=True)
   Deprecated!! Use Matcher instead.
robot.utils.match.matches_any (string, patterns, ignore=(), caseless=True, spaceless=True)
    Deprecated!! Use MultiMatcher instead.

class robot.utils.match.Matcher (pattern, ignore=(), caseless=True, spaceless=True)
    Bases: object
    match (string)

class robot.utils.match.MultiMatcher (patterns=None, ignore=(), caseless=True, spaceless=True, match_if_no_patterns=False)
    Bases: object
    match (string)

4.14.18 misc Module

robot.utils.misc.printable_name (string, code_style=False)
    Generates and returns printable name from the given string.
    If ‘code_style’ is True:
        ‘mixedCAPSCamel’ -> ‘Mixed CAPS Camel’ ‘camelCaseName’ -> ‘Camel Case Name’ ‘under_score_name’ -> ‘Under Score Name’ ‘under_and space’ -> ‘Under And Space’ ‘miXed_CAPS_nAMe’ -> ‘MiXed CAPS NAMe’ ‘’ -> ‘’

robot.utils.misc.plural_or_not (item)

robot.utils.misc.seq2str (sequence, quote='', sep=':', lastsep=' and ')
    Returns sequence in format ‘item 1’, ‘item 2’ and ‘item 3’

robot.utils.misc.seq2str2 (sequence)
    Returns sequence in format [ item 1 | item 2 | ... ]

robot.utils.misc.getdoc (item)

robot.utils.misc.isatty (stream)

4.14.19 normalizing Module

robot.utils.normalizing.normalize (string, ignore=(), caseless=True, spaceless=True)
    Normalizes given string according to given spec.
    By default string is turned to lower case and all whitespace is removed. Additional characters can be removed by giving them in ignore list.

robot.utils.normalizing.lower (string)

robot.utils.normalizing.normalize_tags (tags)
    DEPRECATED!! Use robot.model.Tags instead.
    Returns tags sorted and duplicates, empty, and NONE removed.
    If duplicate tags have different case/space, the one used first wins.

class robot.utils.normalizing.NormalizedDict (initial=None, ignore=(), caseless=True, spaceless=True)
    Bases: UserDict.UserDict
    Custom dictionary implementation automatically normalizing keys.

4.14. utils Package 445
Initializes with possible initial value and normalizing spec.

Initial values can be either a dictionary or an iterable of name/value pairs. In the latter case items are added in the given order.

Normalizing spec has exact same semantics as with `normalize` method.

```python
update(dict=None, **kwargs)
set(key, value)
get(key, default=None)
pop(key)
clear()
has_key(key)
keys()
iterkeys()
values()
itervalues()
items()
iteritems()
copy()
classmethod fromkeys(iterable, value=None)
popitem()
setdefault(key, failobj=None)
```

### 4.14.20 robotenv Module

```python
robot.utils.robotenv.get_env_var(name, default=None)
robot.utils.robotenv.set_env_var(name, value)
robot.utils.robotenv.del_env_var(name)
robot.utils.robotenv.get_env_vars()
```

### 4.14.21 robotpath Module

```python
robot.utils.robotpath.normpath(path)
Returns path in normalized and absolute format.

On case-insensitive file systems the path is also case normalized. If that is not desired, `abspath` should be used instead.

robot.utils.robotpath.abspath(path)
Replacement for os.path.abspath with some bug fixes and enhancements.

1. Converts non-Unicode paths to Unicode using file system encoding
2. At least Jython 2.5.1 on Windows returns wrong path with ‘c:’.
3. Python until 2.6.5 and at least Jython 2.5.1 don’t handle non-ASCII characters in the working directory:
http://bugs.python.org/issue3426

robot.utils.robotpath.get_link_path(target, base)
Returns a relative path to a target from a base.
If base is an existing file, then its parent directory is considered. Otherwise, base is assumed to be a directory.
Rationale: os.path.relpath is not available before Python 2.6

4.14.22 robottime Module

robot.utils.robottime.timestr_to_secs(timestr)
Parses time in format like ‘1h 10s’ and returns time in seconds (float).
Given time must be in format ‘1d 2h 3m 4s 5ms’ with following rules:
• Time parts having zero value can be ignored (e.g. ‘3m 4s’ is ok)
• Format is case and space insensitive
• Instead of ‘d’ it is also possible to use ‘day’ or ‘days’
• Instead of ‘h’ also ‘hour’ and ‘hours’ are ok
• Instead of ‘m’ also ‘minute’, ‘minutes’, ‘min’ and ‘mins’ are ok
• Instead of ‘s’ also ‘second’, ‘seconds’, ‘sec’ and ‘secs’ are ok
• Instead of ‘ms’ also ‘millisecond’, ‘milliseconds’ and ‘millis’ are ok
• It is possible to give time only as a float and then it is considered to be seconds (e.g. ‘123’, ‘123.0’, ‘123s’, ‘2min 3s’ are all equivalent)

robot.utils.robottime.secs_to_timestr(secs, compact=False)
Converts time in seconds to a string representation.
Returned string is in format like ‘1 day 2 hours 3 minutes 4 seconds 5 milliseconds’ with following rules:
• Time parts having zero value are not included (e.g. ‘3 minutes 4 seconds’ instead of ‘0 days 0 hours 3 minutes 4 seconds’)
• Hour part has a maximum of 23 and minutes and seconds both have 59 (e.g. ‘1 minute 40 seconds’ instead of ‘100 seconds’)
If compact has value ‘True’, short suffixes are used. (e.g. 1d 2h 3min 4s 5ms)

robot.utils.robottime.format_time(timetuple_or_epochsecs, daysep='', daytimesep=' ', timesep=':', millissep=None, gmtsep=None)
Returns a timestamp formatted from given time using separators.
Time can be given either as a timetuple or seconds after epoch.
Timetuple is (year, month, day, hour, min, sec[, millis]), where parts must be integers and millis is required only when millisep is not None. Notice that this is not 100% compatible with standard Python timetuples which do not have millis.
Seconds after epoch can be either an integer or a float.

robot.utils.robottime.get_time(format='timestamp', time_=None)
Return the given or current time in requested format.
If time is not given, current time is used. How time is returned is determined based on the given ‘format’ string as follows. Note that all checks are case insensitive.
• If ‘format’ contains word ‘epoch’ the time is returned in seconds after the unix epoch.

• If ‘format’ contains any of the words ‘year’, ‘month’, ‘day’, ‘hour’, ‘min’ or ‘sec’ only selected parts are returned. The order of the returned parts is always the one in previous sentence and order of words in ‘format’ is not significant. Parts are returned as zero padded strings (e.g. May -> ‘05’).

• Otherwise (and by default) the time is returned as a timestamp string in format ‘2006-02-24 15:08:31’

```python
robot.utils.robottime.parse_time(timestr)
```
Parses the time string and returns its value as seconds since epoch.

Time can be given in five different formats:

1. Numbers are interpreted as time since epoch directly. It is possible to use also ints and floats, not only strings containing numbers.

2. Valid timestamp (‘YYYY-MM-DD hh:mm:ss’ and ‘YYYYMMDD hhmmss’).

3. ‘NOW’ (case-insensitive) is the current local time.

4. ‘UTC’ (case-insensitive) is the current time in UTC.

5. Format ‘NOW - 1 day’ or ‘UTC + 1 hour 30 min’ is the current local/UTC time plus/minus the time specified with the time string.

Seconds are rounded down to avoid getting times in the future.

```python
robot.utils.robottime.get_timestamp(daysep='', daytimesep=' ', timesep=':', millissep='.'
robot.utils.robottime.timestamp_to_secs(timestamp, seps=None)
robot.utils.robottime.secs_to_timestamp(secs, seps=None, millis=False)
robot.utils.robottime.get_start_timestamp(daysep='', daytimesep=' ', timesep=':', millissep=None)
robot.utils.robottime.get_elapsed_time(start_time, end_time)
```

Returns the time between given timestamps in milliseconds.

```python
robot.utils.robottime.elapsed_time_to_string(elapsed, include_millis=True)
```

Converts elapsed time in milliseconds to format ‘hh:mm:ss.mil’.

If `include_millis` is True, ‘.mil’ part is omitted.

```python
class robot.utils.robottime.TimestampCache
    Bases: object

    get_timestamp(daysep='', daytimesep=' ', timesep=':', millissep=')
```

### 4.14.23 setter Module

```python
class robot.utils.setter.setter(method)
    Bases: object

class robot.utils.setter.SetterAwareType
    Bases: type

    mro() -> list
    return a type’s method resolution order
```
4.14.24 text Module

robot.utils.text.cut_long_message(msg)
robot.utils.text.format_assign_message(variable, value, cut_long=True)
robot.utils.text.get_console_length(text)
robot.utils.text.pad_console_length(text, width)

4.14.25 unic Module

robot.utils.unic.unic(item, *args)
robot.utils.unic.safe_repr(item)

4.15 variables Package

4.15.1 variables Package

Implements handling and resolving of variables.
This package is likely to change radically in RF 2.8. External code should use functionality provided directly by this package with care.
robot.variables.init_global_variables(settings)

4.15.2 isvar Module

robot.variables.isvar.is_var(string)
robot.variables.isvar.is_scalar_var(string)
robot.variables.isvar.is_list_var(string)

4.15.3 variableassigner Module

class robot.variables.variableassigner.VariableAssigner(assign)
    Bases: object
    assign(context, return_value)

class robot.variables.variableassigner.AssignParser(assign)
    Bases: object

class robot.variables.variableassignerReturnValue(scalar_vars, list_var, return_value)
    Bases: object
    get_variables_to_set()
4.15.4 variables Module

robot.variables.variables.getJavaSystemProperty(name)

class robot.variables.variables.Map

class robot.variables.variables.Variables(identifiers=('$', '@', '%', '&', '*'))
    Bases: robot.utils.normalizing.NormalizedDict

    Represents a set of variables including both $scalars and @lists.
    Contains methods for replacing variables from list, scalars, and strings. On top of $scalar and @list variables these methods handle also %environment variables.

    update(dict=None, **kwargs)

    replace_list(items)
        Replaces variables from a list of items.
        If an item in a list is a @list variable its value is returned. Possible variables from other items are replaced using replace_scalar. Result is always a list.

    replace_scalar(item)
        Replaces variables from a scalar item.
        If the item is not a string it is returned as is. If it is a $scalar variable its value is returned. Otherwise variables are replaced with replace_string. Result may be any object.

    replace_string(string, splitted=None, ignore_errors=False)
        Replaces variables from a string. Result is always a string.

    set_from_file(path, args=None, overwrite=False)

    set_from_variable_table(variable_table, overwrite=False)

    has_key(variable)

    contains(variable, extended=False)

    clear()

    copy()

classmethod fromkeys(iterable, value=None)

    get(key, default=None)

    items()

    iteritems()

    iterkeys()

    itervalues()

    keys()

    pop(key)

    popitem()

    set(key, value)

    setdefault(key, failobj=None)

    values()
4.15.5 variablesplitter Module

class robot.variables.variablesplitter.VariableSplitter(string, identifiers)

    get_replaced_base(variables)

4.16 writer Package

4.16.1 writer Package

Implements writing of parsed test data to files.

This functionality is used by robot.parsing.model.TestCaseFile.save().

This package is considered stable.

4.16.2 aligners Module

class robot.writer.aligners.FirstColumnAligner(first_column_width)
    Bases: robot.writer.aligners._Aligner
    
    align_row(row)

    align_rows(rows)

class robot.writer.aligners.ColumnAligner(first_column_width, table)
    Bases: robot.writer.aligners._Aligner
    
    align_row(row)

    align_rows(rows)

class robot.writer.aligners.NullAligner(widths=None)
    Bases: robot.writer.aligners._Aligner
    
    align_rows(rows)

    align_row(row)

4.16.3 dataextractor Module

class robot.writer.dataextractor.DataExtractor(want_name_on_first_row=None)
    Bases: object
    
    Transforms table of a parsed test data file into a list of rows.

    rows_from_table(table)

4.16.4 datafilewriter Module

class robot.writer.datafilewriter.DataFileWriter(**options)
    Bases: object
    
    Object to write parsed test data file objects back to disk.

    Parameters **options – A WritingContext is created based
on these.

write(datafile)
  Writes given datafile using **options.

Parameters datafile (TestCaseFile, ResourceFile, TestDataDirectory) – The parsed test data object to be written

class robot.writer.datafilewriter.WritingContext(datafile, format='', output=None, pipe_separated=False, txt_separating_spaces=4, line_separator='n')

Bases: object

Contains configuration used in writing a test data file to disk.

Parameters

  * datafile (TestCaseFile, ResourceFile, TestDataDirectory) – The datafile to be written.
  * format (str) – Output file format. If omitted, read from the extension of the source attribute of the given datafile.
  * output – An open, file-like object used in writing. If omitted, value of source attribute of the given datafile is used to construct a new file object.
  * pipe_separated (bool) – Whether to use pipes as separator when output file format is txt.
  * txt_separating_spaces (int) – Number of separating spaces between cells in space separated format.
  * line_separator (str) – Line separator used in output files.

If output is not given, an output file is created based on the source of the given datafile and value of format.

Examples:

Write output in a StringIO instance using format of datafile.source:

WriteConfiguration(datafile, output=StringIO)

Output file is created from datafile.source by stripping extension and replacing it with html:

WriteConfiguration(datafile, format='html')

  encoding = 'UTF-8'
  txt_format = 'txt'
  html_format = 'html'
  tsv_format = 'tsv'
  robot_format = 'robot'
  txt_column_count = 8
  html_column_count = 5
  tsv_column_count = 8

4.16.5 filewriters Module

robot.writer.filewriters.FileWriter(context)
  Creates and returns a FileWriter object.
Parameters context (WritingContext) – Type of returned FileWriter is determined based on context.format. context is also passed to created writer.

class robot.writer.filewriters.SpaceSeparatedTxtWriter (configuration)
    Bases: robot.writer.filewriters._DataFileWriter
    write (datafile)

class robot.writer.filewriters.PipeSeparatedTxtWriter (configuration)
    Bases: robot.writer.filewriters._DataFileWriter
    write (datafile)

class robot.writer.filewriters.TsvFileWriter (configuration)
    Bases: robot.writer.filewriters._DataFileWriter
    write (datafile)

class robot.writer.filewriters.HtmlFileWriter (configuration)
    Bases: robot.writer.filewriters._DataFileWriter
    write (datafile)

4.16.6 formatters Module

class robot.writer.formatters.TsvFormatter (column_count)
    Bases: robot.writer.formatters._DataFileFormatter
    empty_row_after (table)
    format_header (table)
    format_table (table)

class robot.writer.formatters.TxtFormatter (column_count)
    Bases: robot.writer.formatters._DataFileFormatter
    empty_row_after (table)
    format_header (table)
    format_table (table)

class robot.writer.formatters.PipeFormatter (column_count)
    Bases: robot.writer.formatters.TxtFormatter
    empty_row_after (table)
    format_header (table)
    format_table (table)

4.16.7 htmlformatter Module

class robot.writer.htmlformatter.HtmlFormatter (column_count)
    Bases: robot.writer.formatters._DataFileFormatter
    format_header (table)
    empty_row_after (table)
    format_table (table)
class robot.writer.htmlformatter.HtmlCell (content='', attributes=None, tag='td', escape=True)
    Bases: object

class robot.writer.htmlformatter.NameCell (name='', attributes=None)
    Bases: robot.writer.htmlformatter.HtmlCell

class robot.writer.htmlformatter.AnchorNameCell (name, type_)
    Bases: robot.writer.htmlformatter.HtmlCell

class robot.writer.htmlformatter.DocumentationCell (content, span)
    Bases: robot.writer.htmlformatter.HtmlCell

class robot.writer.htmlformatter.HeaderCell (name, span=1)
    Bases: robot.writer.htmlformatter.HtmlCell

4.16.8 htmltemplate Module

4.16.9 rowsplitter Module

class robot.writer.rowsplitter.RowSplitter (cols=8, split_multiline_doc=True)
    Bases: object

    split (row, table_type)
Indices

- genindex
- modindex
- search