Robot Framework Documentation

Release 3.1.2

Robot Framework developers

Nov 15, 2019

Contents

1 Entry points **3**

2 Java entry points **5**

3 Public API **7**

4 All packages **9**
4.1 robot package 9

5 Indices **319**

Python Module Index **321**

Index **325**

This documentation describes the public API of Robot Framework. Installation, basic usage and wealth of other topics are covered by the Robot Framework User Guide.

Main API entry points are documented here, but the lower level implementation details are not always 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 robotframework-users mailing list.

CHAPTER 1

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 (Rebot).
- `robot.libdoc` entry point for Libdoc tool.
- `robot.testdoc` entry point for Testdoc tool.
- `robot.tidy` entry point for Tidy tool.

See built-in tool documentation for more details about Rebot, Libdoc, Testdoc, and Tidy tools.

CHAPTER 2

Java entry points

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

CHAPTER 3

Public API

robot.api package exposes the public APIs of Robot Framework.

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

Currently exposed APIs are:

- *logger* module for test libraries' logging purposes.
- *deco* module with decorators test libraries can utilize.
- *TestCaseFile*, *TestDataDirectory*, and *ResourceFile* classes for parsing test data files and directories. In addition, a convenience factory method *TestData()* creates either *TestCaseFile* or *TestDataDirectory* objects based on the input.
- *TestSuite* class for creating executable test suites programmatically and *TestSuiteBuilder* class for creating such suites based on existing test data on the file system.
- *SuiteVisitor* abstract class for processing testdata before execution. This can be used as a base for implementing a pre-run modifier that is taken into use with `--prerunmodifier` commandline option.
- *ExecutionResult()* factory method for reading execution results from XML output files and *ResultVisitor* abstract class to ease further processing the results. *ResultVisitor* can also be used as a base for pre-Rebot modifier that is taken into use with `--prerebotmodifier` commandline option.
- *ResultWriter* class for writing reports, logs, XML outputs, and XUnit files. Can write results based on XML outputs on the file system, as well as based on the result objects returned by the *ExecutionResult()* or an executed *TestSuite*.

All of the above names can be imported like:

```
from robot.api import ApiName
```

See documentations of the individual APIs for more details.

Tip: APIs related to the command line entry points are exposed directly via the *robot* root package.

CHAPTER 4

All packages

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

The root of the Robot Framework package.

The command line entry points provided by the framework are exposed for programmatic usage as follows:

- *run()*: Function to run tests.
- *run_cli()*: Function to run tests with command line argument processing.
- *rebot()*: Function to post-process outputs.
- *rebot_cli()*: Function to post-process outputs with command line argument processing.
- *libdoc*: Module for library documentation generation.
- *testdoc*: Module for test case documentation generation.
- *tidy*: Module for test data clean-up and format change.

All the functions above can be imported like `from robot import run`. Functions and classes provided by the modules need to be imported like `from robot.libdoc import libdoc_cli`.

The functions and modules listed above are considered stable. Other modules in this package are for for internal usage and may change without prior notice.

Tip: More public APIs are exposed by the *robot.api* package.

robot.**run**(**tests*, ***options*)

Programmatic entry point for running tests.

Parameters

- **tests** – Paths to test case files/directories to be executed similarly as when running the `robot` command on the command line.
- **options** – Options to configure and control execution. Accepted options are mostly same as normal command line options to the `robot` command. Option names match command line option long names without hyphens so that, for example, `--name` becomes `name`.

Most options that can be given from the command line work. An exception is that options `--pythonpath`, `--argumentfile`, `--help` and `--version` are not supported.

Options that can be given on the command line multiple times can be passed as lists. For example, `include=['tag1', 'tag2']` is equivalent to `--include tag1 --include tag2`. If such options are used only once, they can be given also as a single string like `include='tag'`.

Options that accept no value can be given as Booleans. For example, `dryrun=True` is same as using the `--dryrun` option.

Options that accept string `NONE` as a special value can also be used with Python `None`. For example, using `log=None` is equivalent to `--log NONE`.

`listener`, `prerunmodifier` and `prerebotmodifier` options allow passing values as Python objects in addition to module names these command line options support. For example, `run('tests', listener=MyListener())`.

To capture the standard output and error streams, pass an open file or file-like object as special keyword arguments `stdout` and `stderr`, respectively.

A return code is returned similarly as when running on the command line. Zero means that tests were executed and no critical test failed, values up to 250 denote the number of failed critical tests, and values between 251-255 are for other statuses documented in the Robot Framework User Guide.

Example:

```
from robot import run

run('path/to/tests.robot')
run('tests.robot', include=['tag1', 'tag2'], splitlog=True)
with open('stdout.txt', 'w') as stdout:
    run('t1.robot', 't2.robot', name='Example', log=None, stdout=stdout)
```

Equivalent command line usage:

```
robot path/to/tests.robot
robot --include tag1 --include tag2 --splitlog tests.robot
robot --name Example --log NONE t1.robot t2.robot > stdout.txt
```

robot.**run_cli**(*arguments=None*, *exit=True*)

Command line execution entry point for running tests.

Parameters

- **arguments** – Command line options and arguments as a list of strings. Starting from RF 3.1, defaults to `sys.argv[1:]` if not given.
- **exit** – If `True`, call `sys.exit` with the return code denoting execution status, otherwise just return the rc. New in RF 3.0.1.

Entry point used when running tests from the command line, but can also be used by custom scripts that execute tests. Especially useful if the script itself needs to accept same arguments as accepted by Robot Framework, because the script can just pass them forward directly along with the possible default values it sets itself.

Example:

```
from robot import run_cli

# Run tests and return the return code.
rc = run_cli(['--name', 'Example', 'tests.robot'], exit=False)

# Run tests and exit to the system automatically.
run_cli(['--name', 'Example', 'tests.robot'])
```

See also the *run()* function that allows setting options as keyword arguments like `name="Example"` and generally has a richer API for programmatic test execution.

`robot.`**`rebot`**(**outputs*, ***options*)

Programmatic entry point for post-processing outputs.

Parameters

- **`outputs`** – Paths to Robot Framework output files similarly as when running the `rebot` command on the command line.
- **`options`** – Options to configure processing outputs. Accepted options are mostly same as normal command line options to the `rebot` command. Option names match command line option long names without hyphens so that, for example, `--name` becomes `name`.

The semantics related to passing options are exactly the same as with the *run()* function. See its documentation for more details.

Examples:

```
from robot import rebot

rebot('path/to/output.xml')
with open('stdout.txt', 'w') as stdout:
    rebot('o1.xml', 'o2.xml', name='Example', log=None, stdout=stdout)
```

Equivalent command line usage:

```
rebot path/to/output.xml
rebot --name Example --log NONE o1.xml o2.xml > stdout.txt
```

`robot.`**`rebot_cli`**(*arguments=None*, *exit=True*)

Command line execution entry point for post-processing outputs.

Parameters

- **`arguments`** – Command line options and arguments as a list of strings. Starting from RF 3.1, defaults to `sys.argv[1:]` if not given.
- **`exit`** – If `True`, call `sys.exit` with the return code denoting execution status, otherwise just return the rc. New in RF 3.0.1.

Entry point used when post-processing outputs from the command line, but can also be used by custom scripts. Especially useful if the script itself needs to accept same arguments as accepted by Rebot, because the script can just pass them forward directly along with the possible default values it sets itself.

Example:

```
from robot import rebot_cli

rebot_cli(['--name', 'Example', '--log', 'NONE', 'o1.xml', 'o2.xml'])
```

See also the *rebot()* function that allows setting options as keyword arguments like `name="Example"` and generally has a richer API for programmatic Rebot execution.

4.1.1 Subpackages

robot.api package

robot.api package exposes the public APIs of Robot Framework.

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

Currently exposed APIs are:

- *logger* module for test libraries' logging purposes.
- *deco* module with decorators test libraries can utilize.
- *TestCaseFile*, *TestDataDirectory*, and *ResourceFile* classes for parsing test data files and directories. In addition, a convenience factory method *TestData()* creates either *TestCaseFile* or *TestDataDirectory* objects based on the input.
- *TestSuite* class for creating executable test suites programmatically and *TestSuiteBuilder* class for creating such suites based on existing test data on the file system.
- *SuiteVisitor* abstract class for processing testdata before execution. This can be used as a base for implementing a pre-run modifier that is taken into use with `--prerunmodifier` commandline option.
- *ExecutionResult()* factory method for reading execution results from XML output files and *ResultVisitor* abstract class to ease further processing the results. *ResultVisitor* can also be used as a base for pre-Rebot modifier that is taken into use with `--prerebotmodifier` commandline option.
- *ResultWriter* class for writing reports, logs, XML outputs, and XUnit files. Can write results based on XML outputs on the file system, as well as based on the result objects returned by the *ExecutionResult()* or an executed *TestSuite*.

All of the above names can be imported like:

```
from robot.api import ApiName
```

See documentations of the individual APIs for more details.

Tip: APIs related to the command line entry points are exposed directly via the *robot* root package.

Submodules

robot.api.deco module

`robot.api.deco.`**`keyword`**(*name=None*, *tags=()*, *types=()*)

Decorator to set custom name, tags and argument types to keywords.

This decorator creates `robot_name`, `robot_tags` and `robot_types` attributes on the decorated keyword method or function based on the provided arguments. Robot Framework checks them to determine the keyword's name, tags, and argument types, respectively.

Name must be given as a string, tags as a list of strings, and types either as a dictionary mapping argument names to types or as a list (or tuple) of types mapped to arguments based on position. It is OK to specify types only to some arguments, and setting `types` to `None` disables type conversion altogether.

Examples:

```
@keyword(name='Login Via User Panel')
def login(username, password):
    # ...

@keyword(name='Logout Via User Panel', tags=['example', 'tags'])
def logout():
    # ...

@keyword(types={'length': int, 'case_insensitive': bool})
def types_as_dict(length, case_insensitive=False):
    # ...

@keyword(types=[int, bool])
def types_as_list(length, case_insensitive=False):
    # ...

@keyword(types=None)
def no_conversion(length, case_insensitive=False):
    # ...
```

If `name` is not given, the actual name of the keyword will not be affected, but the `robot_name` attribute will still be created. This can be useful for marking methods as keywords in a dynamic library. In this usage it is possible to also omit parenthesis when using the decorator:

```
@keyword
def func():
    # ...
```

robot.api.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.

If the logging methods are used when Robot Framework is not running, the messages are redirected to the standard Python `logging` module using logger named `RobotFramework`.

Log levels

It is possible to log messages using levels TRACE, DEBUG, INFO, WARN and ERROR either using the *write()* function or, more commonly, with the log level specific *trace()*, *debug()*, *info()*, *warn()*, *error()* functions. The support for the error level and function is new in RF 2.9.

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

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. Alternatively, *write()* accepts a pseudo log level HTML.

Example

```
from robot.api import logger

def my_keyword(arg):
    logger.debug('Got argument %s.' % arg)
    do_something()
    logger.info('<i>This</i> is a boring example.', html=True)
```

robot.api.logger.**write**(*msg*, *level='INFO'*, *html=False*)
Writes the message to the log file using the given level.

Valid log levels are TRACE, DEBUG, INFO (default since RF 2.9.1), WARN, and ERROR (new in RF 2.9). Additionally it is possible to use HTML pseudo log level that logs the message as HTML using the INFO level.

Instead of using this method, it is generally better to use the level specific methods such as info and debug that have separate html argument to control the message format.

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

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

robot.api.logger.**info**(*msg*, *html=False*, *also_console=False*)
Writes the message to the log file using 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 using the WARN level.

robot.api.logger.**error**(*msg*, *html=False*)
Writes the message to the log file using the ERROR level.

New in Robot Framework 2.9.

robot.api.logger.**console**(*msg*, *newline=True*, *stream='stdout'*)
Writes the message to the console.

If the newline argument is True, a newline character is automatically added to the message.

By default the message is written to the standard output stream. Using the standard error stream is possibly by giving the `stream` argument value `'stderr'`.

robot.conf package

Implements settings for both test execution and output processing.

This package implements *`RobotSettings`* and *`RebotSettings`* classes used internally by the framework. There should be no need to use these classes externally.

This package can be considered relatively stable. Aforementioned classes are likely to be rewritten at some point to be more convenient to use. Instantiating them is not likely to change, though.

Submodules

robot.conf.gatherfailed module

class robot.conf.gatherfailed.**GatherFailedTests**
Bases: *`robot.model.visitor.SuiteVisitor`*

visit_test(*test*)
Implements traversing through the test and its keywords.

Can be overridden to allow modifying the passed in `test` without calling *`start_test()`* or *`end_test()`* nor visiting keywords.

visit_keyword(*kw*)
Implements traversing through the keyword and its child keywords.

Can be overridden to allow modifying the passed in `kw` without calling *`start_keyword()`* or *`end_keyword()`* nor visiting child keywords.

end_keyword(*keyword*)
Called when keyword ends. Default implementation does nothing.

end_message(*msg*)
Called when message ends. Default implementation does nothing.

end_suite(*suite*)
Called when suite ends. Default implementation does nothing.

end_test(*test*)
Called when test ends. Default implementation does nothing.

start_keyword(*keyword*)
Called when keyword starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_message(*msg*)
Called when message starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_suite(*suite*)
Called when suite starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_test(*test*)
Called when test starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

visit_message(*msg*)
Implements visiting the message.

Can be overridden to allow modifying the passed in `msg` without calling *start_message()* or *end_message()*.

visit_suite(*suite*)
Implements traversing through the suite and its direct children.

Can be overridden to allow modifying the passed in `suite` without calling *start_suite()* or *end_suite()* nor visiting child suites, tests or keywords (setup and teardown) at all.

class robot.conf.gatherfailed.**GatherFailedSuites**
Bases: *robot.model.visitor.SuiteVisitor*

start_suite(*suite*)
Called when suite starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

visit_test(*test*)
Implements traversing through the test and its keywords.

Can be overridden to allow modifying the passed in `test` without calling *start_test()* or *end_test()* nor visiting keywords.

visit_keyword(*kw*)
Implements traversing through the keyword and its child keywords.

Can be overridden to allow modifying the passed in `kw` without calling *start_keyword()* or *end_keyword()* nor visiting child keywords.

end_keyword(*keyword*)
Called when keyword ends. Default implementation does nothing.

end_message(*msg*)
Called when message ends. Default implementation does nothing.

end_suite(*suite*)
Called when suite ends. Default implementation does nothing.

end_test(*test*)
Called when test ends. Default implementation does nothing.

start_keyword(*keyword*)
Called when keyword starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_message(*msg*)
Called when message starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_test(*test*)
Called when test starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

visit_message(*msg*)
Implements visiting the message.

Can be overridden to allow modifying the passed in `msg` without calling *start_message()* or *end_message()*.

visit_suite(*suite*)
Implements traversing through the suite and its direct children.

Can be overridden to allow modifying the passed in `suite` without calling *start_suite()* or *end_suite()* nor visiting child suites, tests or keywords (setup and teardown) at all.

robot.conf.gatherfailed.**gather_failed_tests**(*output*)

robot.conf.gatherfailed.**gather_failed_suites**(*output*)

robot.conf.settings module

class robot.conf.settings.**RobotSettings**(*options=None*, ***extra_options*)
Bases: robot.conf.settings._BaseSettings

get_rebot_settings()

listeners

debug_file

suite_config

randomize_seed

randomize_suites

randomize_tests

dry_run

exit_on_failure

exit_on_error

skip_teardown_on_exit

console_output_config

console_type

console_width

console_markers

max_error_lines

pre_run_modifiers

run_empty_suite

variables

variable_files

extension

console_colors

critical_tags

flatten_keywords

log

log_level

non_critical_tags

output

output_directory

pre_rebot_modifiers

remove_keywords

report

rpa

split_log

statistics_config

status_rc

xunit

xunit_skip_noncritical

class robot.conf.settings.**RebotSettings**(*options=None*, ***extra_options*)

Bases: robot.conf.settings._BaseSettings

suite_config

log_config

report_config

merge

console_colors

console_output_config

critical_tags

flatten_keywords

log

log_level

non_critical_tags

output

output_directory

pre_rebot_modifiers

remove_keywords

report

rpa

split_log

statistics_config

status_rc

xunit

xunit_skip_noncritical

process_empty_suite

robot.htmldata package

Package for writing output files in HTML format.

This package is considered stable but it is not part of the public API.

Submodules

robot.htmldata.htmlfilewriter 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*)

robot.htmldata.jartemplate module

robot.htmldata.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*)

robot.htmldata.normaltemplate module

class robot.htmldata.normaltemplate.**HtmlTemplate**(*filename*)
Bases: object

robot.htmldata.template module

robot.libdocpkg package

Implements the *Libdoc* tool.

The command line entry point and programmatic interface for Libdoc are provided by the separate *robot.libdoc* module.

This package is considered stable but it is not part of the public API.

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

Submodules

robot.libdocpkg.builder module

robot.libdocpkg.builder.**JavaDocBuilder**()

robot.libdocpkg.builder.**DocumentationBuilder**(*library_or_resource*)

robot.libdocpkg.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*)

robot.libdocpkg.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**(*doc_format*)
Bases: object

robot.libdocpkg.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. Returned object implements com.sun.javadoc.ClassDoc interface: http://docs.oracle.com/javase/7/docs/jdk/api/javadoc/doclet/

robot.libdocpkg.model module

class robot.libdocpkg.model.**LibraryDoc**(*name=''*, *doc=''*, *version=''*, *type='library'*, *scope=''*, *named_args=True*, *doc_format=''*)
Bases: object

doc_format

keywords

all_tags

save(*output=None*, *format='HTML'*)

class robot.libdocpkg.model.**KeywordDoc**(*name=''*, *args=()*, *doc=''*, *tags=()*)
Bases: *robot.utils.sortable.Sortable*

shortdoc

robot.libdocpkg.output module

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

robot.libdocpkg.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**(*resource=False*)
Bases: object

build_keywords(*lib*)

build_keyword(*kw*)

robot.libdocpkg.specbuilder module

class robot.libdocpkg.specbuilder.**SpecDocBuilder**
Bases: object

build(*path*)

robot.libdocpkg.writer module

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

robot.libdocpkg.xmlwriter module

class robot.libdocpkg.xmlwriter.**LibdocXmlWriter**
Bases: object

write(*libdoc*, *outfile*)

robot.libraries package

Package hosting Robot Framework standard test libraries.

Libraries are mainly used externally in the test data, but they can be also used by custom test libraries if there is a need. Especially the *BuiltIn* library is often useful when there is a need to interact with the framework.

Because libraries are documented using Robot Framework's own documentation syntax, the generated API docs are not that well formed. It is thus better to find the generated library documentations, for example, via the http://robotframework.org web site.

Submodules

robot.libraries.BuiltIn module

robot.libraries.BuiltIn.**run_keyword_variant**(*resolve*)

class robot.libraries.BuiltIn.**BuiltIn**
Bases: robot.libraries.BuiltIn._Verify, robot.libraries.BuiltIn._Converter, robot.libraries.BuiltIn._Variables, robot.libraries.BuiltIn._RunKeyword, robot.libraries.BuiltIn._Control, 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*).

== Table of contents ==

- *HTML error messages*
- *Evaluating expressions*
- *Boolean arguments*
- *Pattern matching*
- *Multiline string comparison*
- *String representations*
- *Shortcuts*
- *Keywords*

= HTML error messages =

Many of the keywords accept an optional error message to use if the keyword fails, and it is possible to use HTML in these messages by prefixing them with *HTML*. See *Fail* keyword for a usage example. Notice that using HTML in messages is not limited to BuiltIn library but works with any error message.

= Evaluating expressions =

Many keywords, such as *Evaluate*, *Run Keyword If* and *Should Be True*, accept an expression that is evaluated in Python. These expressions are evaluated using Python's [http://docs.python.org/library/functions.html#eval|eval] function so that all Python built-ins like len() and int() are available. *Evaluate* allows configuring the execution namespace with custom modules, and other keywords have [http://docs.python.org/library/os.html|os] and [http://docs.python.org/library/sys.html|sys] modules available automatically.

When a variable is used in the expressing using the normal ${variable} syntax, its value is replaces before the expression is evaluated. This means that the value used in the expression will be the string representation of the variable value, not the variable value itself. This is not a problem with numbers and other objects that have a string representation that can be evaluated directly, but with other objects the behavior depends on the string representation. Most importantly, strings must always be quoted, and if they can contain newlines, they must be triple quoted.

Starting from Robot Framework 2.9, variables themselves are automatically available in the evaluation namespace. They can be accessed using special variable syntax without the curly braces like $variable. These variables should never be quoted, and in fact they are not even replaced inside strings.

Using the $variable syntax slows down expression evaluation a little. This should not typically matter, but should be taken into account if complex expressions are evaluated often and there are strict time constrains.

Notice that instead of creating complicated expressions, it is often better to move the logic into a test library. That eases maintenance and can also enhance execution speed.

= Boolean arguments =

Some keywords accept arguments that are handled as Boolean values true or false. If such an argument is given as a string, it is considered false if it is an empty string or equal to FALSE, NONE, NO, OFF or 0, case-insensitively. Keywords verifying something that allow dropping actual and expected values from the possible error message also consider string no values to be false. Other strings are considered true unless the keyword documentation explicitly states otherwise, and other argument types are tested using the same [http://docs.python.org/library/stdtypes.html#truth|rules as in Python].

True examples:

False examples:

Considering string `NONE` false is new in Robot Framework 3.0.3 and considering also `OFF` and `0` false is new in Robot Framework 3.1.

= Pattern matching =

Many keywords accepts arguments as either glob or regular expression patterns.

== Glob patterns ==

Some keywords, for example *Should Match*, support so called [http://en.wikipedia.org/wiki/Glob_(programming)|glob patterns] where:

Unlike with glob patterns normally, path separator characters `/` and `\` and the newline character `\n` are matches by the above wildcards.

Support for brackets like `[abc]` and `[!a-z]` is new in Robot Framework 3.1

== Regular expressions ==

Some keywords, for example *Should Match Regexp*, support [http://en.wikipedia.org/wiki/Regular_expression|regular expressions] that are more powerful but also more complicated that glob patterns. The regular expression support is implemented using Python's [http://docs.python.org/library/re.html|re module] and its documentation should be consulted for more information about the syntax.

Because the backslash character (`\`) is an escape character in Robot Framework test data, possible backslash characters in regular expressions need to be escaped with another backslash like `\\d\\w+`. Strings that may contain special characters but should be handled as literal strings, can be escaped with the *Regexp Escape* keyword.

= Multiline string comparison =

Should Be Equal and *Should Be Equal As Strings* report the failures using [http://en.wikipedia.org/wiki/Diff_utility#Unified_format|unified diff format] if both strings have more than two lines. New in Robot Framework 2.9.1.

Results in the following error message:

= String representations =

Several keywords log values explicitly (e.g. *Log*) or implicitly (e.g. *Should Be Equal* when there are failures). By default keywords log values using "human readable" string representation, which means that strings like `Hello` and numbers like `42` are logged as-is. Most of the time this is the desired behavior, but there are some problems as well:

- It is not possible to see difference between different objects that have same string representation like string `42` and integer `42`. *Should Be Equal* and some other keywords add the type information to the error message in these cases, though.
- Non-printable characters such as the null byte are not visible.
- Trailing whitespace is not visible.
- Different newlines (`\r\n` on Windows, `\n` elsewhere) cannot be separated from each others.
- There are several Unicode characters that are different but look the same. One example is the Latin `a` (`\u0061`) and the Cyrillic `а` (`\u0430`). Error messages like `a != а` are not very helpful.
- Some Unicode characters can be represented using [https://en.wikipedia.org/wiki/Unicode_equivalence|different forms]. For example, `ä` can be represented either as a single code point `\u00e4` or using two code points `\u0061` and `\u0308` combined together. Such forms are

considered canonically equivalent, but strings containing them are not considered equal when compared in Python. Error messages like `ä != a` are not that helpful either.

- Containers such as lists and dictionaries are formatted into a single line making it hard to see individual items they contain.

To overcome the above problems, some keywords such as *Log* and *Should Be Equal* have an optional `formatter` argument that can be used to configure the string representation. The supported values are `str` (default), `repr`, and `ascii` that work similarly as [https://docs.python.org/library/functions.html|Python built-in functions] with same names. More detailed semantics are explained below.

The `formatter` argument is new in Robot Framework 3.1.2.

== str ==

Use the "human readable" string representation. Equivalent to using `str()` in Python 3 and `unicode()` in Python 2. This is the default.

== repr ==

Use the "machine readable" string representation. Similar to using `repr()` in Python, which means that strings like `Hello` are logged like `'Hello'`, newlines and non-printable characters are escaped like `\n` and `\x00`, and so on. Non-ASCII characters are shown as-is like `ä` in Python 3 and in escaped format like `\xe4` in Python 2. Use `ascii` to always get the escaped format.

There are also some enhancements compared to the standard `repr()`: - Bigger lists, dictionaries and other containers are pretty-printed so

that there is one item per row.

- On Python 2 the `u` prefix is omitted with Unicode strings and the `b` prefix is added to byte strings.

== ascii ==

Same as using `ascii()` in Python 3 or `repr()` in Python 2 where `ascii()` does not exist. Similar to using `repr` explained above but with the following differences:

- On Python 3 non-ASCII characters are escaped like `\xe4` instead of showing them as-is like `ä`. This makes it easier to see differences between Unicode characters that look the same but are not equal. This is how `repr()` works in Python 2.
- On Python 2 just uses the standard `repr()` meaning that Unicode strings get the `u` prefix and no `b` prefix is added to byte strings.
- Containers are not pretty-printed.

ROBOT_LIBRARY_SCOPE = 'GLOBAL'

ROBOT_LIBRARY_VERSION = '3.1.2'

call_method(*object*, *method_name*, **args*, ***kwargs*)
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.

Support for `**kwargs` is new in Robot Framework 2.9. Since that possible equal signs in other arguments must be escaped with a backslash like `\=`.

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 instead. Items are converted into strings when necessary.

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.

continue_for_loop()
Skips the current for loop iteration and continues from the next.

Skips the remaining keywords in the current for loop iteration and continues from the next one. Can be used directly in a for loop or in a keyword that the loop uses.

See *Continue For Loop If* to conditionally continue a for loop without using *Run Keyword If* or other wrapper keywords.

continue_for_loop_if (*condition*)
Skips the current for loop iteration if the `condition` is true.

A wrapper for *Continue For Loop* to continue a for loop based on the given condition. The condition is evaluated using the same semantics as with *Should Be True* keyword.

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.

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 [http://docs.python.org/library/stdtypes.html#truth|truth value] using Python's `bool()` method.

convert_to_bytes (*input*, *input_type='text'*)
Converts the given `input` to bytes according to the `input_type`.

Valid input types are listed below:

- `text`: Converts text to bytes character by character. All characters with ordinal below 256 can be used and are converted to bytes with same values. Many characters are easiest to represent using escapes like `\x00` or `\xff`. Supports both Unicode strings and bytes.
- `int`: Converts integers separated by spaces to bytes. Similarly as with *Convert To Integer*, it is possible to use binary, octal, or hex values by prefixing the values with `0b`, `0o`, or `0x`, respectively.
- `hex`: Converts hexadecimal values to bytes. Single byte is always two characters long (e.g. `01` or `FF`). Spaces are ignored and can be used freely as a visual separator.
- `bin`: Converts binary values to bytes. Single byte is always eight characters long (e.g. `00001010`). Spaces are ignored and can be used freely as a visual separator.

In addition to giving the input as a string, it is possible to use lists or other iterables containing individual characters or numbers. In that case numbers do not need to be padded to certain length and they cannot contain extra spaces.

Use *Encode String To Bytes* in `String` library if you need to convert text to bytes using a certain encoding.

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 the `lowercase` argument a true value (see *Boolean arguments*) turns the value (but not the given prefix) to lower case.

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. There are two ways to convert from other bases:

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

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

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. If a number is equally close to a certain precision, it is always rounded away from zero.

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, these resources:

- http://docs.python.org/tutorial/floatingpoint.html
- http://randomascii.wordpress.com/2012/02/25/comparing-floating-point-numbers-2012-edition

If you want to avoid possible problems with floating point numbers, you can implement custom keywords using Python's [http://docs.python.org/library/decimal.html|decimal] or [http://docs.python.org/library/fractions.html|fractions] modules.

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.

See also *Convert To Integer*, *Convert To Binary* and *Convert To Hex*.

convert_to_string(*item*)
Converts the given item to a Unicode string.

Strings are also [http://www.macchiato.com/unicode/nfc-faq| NFC normalized].

Use *Encode String To Bytes* and *Decode Bytes To String* keywords in `String` library if you need to convert between Unicode and byte strings using different encodings. Use *Convert To Bytes* if you just want to create byte strings.

create_dictionary(**items*)
Creates and returns a dictionary based on the given `items`.

Items are typically given using the `key=value` syntax same way as `&{dictionary}` variables are created in the Variable table. Both keys and values can contain variables, and possible equal sign in key can be escaped with a backslash like `escaped\=key=value`. It is also possible to get items from existing dictionaries by simply using them like `&{dict}`.

Alternatively items can be specified so that keys and values are given separately. This and the `key=value` syntax can even be combined, but separately given items must be first. If same key is used multiple times, the last value has precedence.

The returned dictionary is ordered, and values with strings as keys can also be accessed using a convenient dot-access syntax like `${dict.key}`. Technically the returned dictionary is Robot Framework's own `DotDict` instance. If there is a need, it can be converted into a regular Python `dict` instance by using the *Convert To Dictionary* keyword from the Collections library.

This keyword was changed in Robot Framework 2.9 in many ways: - Moved from the Collections library to BuiltIn. - Support also non-string keys in `key=value` syntax. - Returned dictionary is ordered and dot-accessible (i.e. `DotDict`). - Old syntax to give keys and values separately was deprecated, but

deprecation was later removed in RF 3.0.1.

create_list(**items*)
Returns a list containing given items.

The returned list can be assigned both to `${scalar}` and `@{list}` variables.

evaluate(*expression*, *modules=None*, *namespace=None*)
Evaluates the given expression in Python and returns the results.

`expression` is evaluated in Python as explained in *Evaluating expressions*.

`modules` argument can be used to specify a comma separated list of Python modules to be imported and added to the evaluation namespace.

`namespace` argument can be used to pass a custom evaluation namespace as a dictionary. Possible `modules` are added to this namespace.

Variables used like `${variable}` are replaced in the expression before evaluation. Variables are also available in the evaluation namespace and can be accessed using special syntax `$variable`. This is a new feature in Robot Framework 2.9 and it is explained more thoroughly in *Evaluating expressions*.

exit_for_loop()
Stops executing the enclosing for loop.

Exits the enclosing for loop and continues execution after it. Can be used directly in a for loop or in a keyword that the loop uses.

See *Exit For Loop If* to conditionally exit a for loop without using *Run Keyword If* or other wrapper keywords.

exit_for_loop_if(*condition*)
Stops executing the enclosing for loop if the `condition` is true.

A wrapper for *Exit For Loop* to exit a for loop based on the given condition. The condition is evaluated using the same semantics as with *Should Be True* keyword.

fail(*msg=None*, **tags*)
Fails the test with the given message and optionally alters its tags.

The error message is specified using the `msg` argument. It is possible to use HTML in the given error message, similarly as with any other keyword accepting an error message, by prefixing the error with `*HTML*`.

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.

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.

get_length(*item*)
Returns and logs the length of the given item as an integer.

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.

See also *Length Should Be*, *Should Be Empty* and *Should Not Be Empty*.

get_library_instance(*name=None*, *all=False*)
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:

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.

If the optional argument `all` is given a true value, then a dictionary mapping all library names to instances will be returned. This feature is new in Robot Framework 2.9.2.

get_time(*format='timestamp'*, *time_='NOW'*)
Returns the given time in the requested format.

NOTE: DateTime library contains much more flexible keywords for getting the current date and time and for date and time handling in general.

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 hhmmss`.
3) If `time` is equal to `NOW` (default), the current local time is used.
4) If `time` is equal to `UTC`, the current time in [http://en.wikipedia.org/wiki/Coordinated_Universal_Time|UTC] is used.
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.

UTC time is 2006-03-29 12:06:21):

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*.

See *Set Variable If* for another keyword to set variables dynamically.

get_variables(*no_decoration=False*)

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.

By default variables are returned with `${}`, `@{}` or `&{}` decoration based on variable types. Giving a true value (see *Boolean arguments*) to the optional argument `no_decoration` will return the variables without the decoration. This option is new in Robot Framework 2.9.

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 paths are used, they must be given in absolute format or found from [http://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#pythonpath-jythonpath-and-ironpythonpath| search path]. Forward slashes can be used as path separators in all operating systems.

It is possible to pass arguments to the imported library and also named argument syntax works if the library supports it. `WITH NAME` syntax can be used to give a custom name to the imported library.

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 or found from [http://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#pythonpath-jythonpath-and-ironpythonpath| search path]. Forward slashes can be used as path separator regardless the operating system.

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. This functionality can thus be used to import new variables, for example, for each test in a test suite.

The given path must be absolute or found from [http://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#pythonpath-jythonpath-and-ironpythonpath| search path]. Forward slashes can be used as path separator regardless the operating system.

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.

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'*, *html=False*, *console=False*, *repr=False*, *formatter='str'*)
Logs the given message with the given level.

Valid levels are TRACE, DEBUG, INFO (default), HTML, WARN, and ERROR. Messages below the current active log level are ignored. See *Set Log Level* keyword and `--loglevel` command line option for more details about setting the level.

Messages logged with the WARN or ERROR levels will be automatically visible also in the console and in the Test Execution Errors section in the log file.

If the `html` argument is given a true value (see *Boolean arguments*), the message will be considered HTML and special characters such as < are not escaped. For example, logging `<img src="image.png">` creates an image when `html` is true, but otherwise the message is that exact string. An alternative to using the `html` argument is using the HTML pseudo log level. It logs the message as HTML using the INFO level.

If the `console` argument is true, the message will be written to the console where test execution was started from in addition to the log file. This keyword always uses the standard output stream and adds a newline after the written message. Use *Log To Console* instead if either of these is undesirable,

The `formatter` argument controls how to format the string representation of the message. Possible values are `str` (default), `repr` and `ascii`, and they work similarly to Python built-in functions with same names. When using `repr`, bigger lists, dictionaries and other containers are also pretty-printed so that there is one item per row. For more details see *String representations*. This is a new feature in Robot Framework 3.1.2.

The old way to control string representation was using the `repr` argument, and `repr=True` is still equivalent to using `formatter=repr`. The `repr` argument will be deprecated in the future, though, and using `formatter` is thus recommended.

See *Log Many* if you want to log multiple messages in one go, and *Log To Console* if you only want to write to the console.

log_many(**messages*)
Logs the given messages as separate entries using the INFO level.

Supports also logging list and dictionary variable items individually.

See *Log* and *Log To Console* keywords if you want to use alternative log levels, use HTML, or log to the console.

log_to_console(*message*, *stream='STDOUT'*, *no_newline=False*)
Logs the given message to the console.

By default uses the standard output stream. Using the standard error stream is possibly by giving the `stream` argument value `STDERR` (case-insensitive).

By default appends a newline to the logged message. This can be disabled by giving the `no_newline` argument a true value (see *Boolean arguments*).

This keyword does not log the message to the normal log file. Use *Log* keyword, possibly with argument `console`, if that is desired.

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

no_operation()
Does absolutely nothing.

pass_execution(*message*, **tags*)
Skips rest of the current test, setup, or teardown with PASS status.

This keyword can be used anywhere in the test data, but the place where used affects the behavior:

- When used in any setup or teardown (suite, test or keyword), passes that setup or teardown. Possible keyword teardowns of the started keywords are executed. Does not affect execution or statuses otherwise.
- When used in a test outside setup or teardown, passes that particular test case. Possible test and keyword teardowns are executed.

Possible continuable failures before this keyword is used, as well as failures in executed teardowns, will fail the execution.

It is mandatory to give a message explaining why execution was passed. By default the message is considered plain text, but starting it with `*HTML*` allows using HTML formatting.

It is also possible to modify test tags passing tags after the message similarly as with *Fail* keyword. 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.

This keyword is typically wrapped to some other keyword, such as *Run Keyword If*, to pass based on a condition. The most common case can be handled also with *Pass Execution If*:

Passing execution in the middle of a test, setup or teardown should be used with care. In the worst case it leads to tests that skip all the parts that could actually uncover problems in the tested application. In cases where execution cannot continue do to external factors, it is often safer to fail the test case and make it non-critical.

pass_execution_if(*condition*, *message*, **tags*)
Conditionally skips rest of the current test, setup, or teardown with PASS status.

A wrapper for *Pass Execution* to skip rest of the current test, setup or teardown based the given `condition`. The condition is evaluated similarly as with *Should Be True* keyword, and `message` and `*tags` have same semantics as with *Pass Execution*.

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.

reload_library(*name_or_instance*)
Rechecks what keywords the specified library provides.

Can be called explicitly in the test data or by a library itself when keywords it provides have changed.

The library can be specified by its name or as the active instance of the library. The latter is especially useful if the library itself calls this keyword as a method.

New in Robot Framework 2.9.

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 with `*`, `?` and `[chars]` acting as wildcards. See the *Glob patterns* section for more information.

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}`.

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(*repeat*, *name*, **args*)
Executes the specified keyword multiple times.

`name` and `args` define the keyword that is executed similarly as with *Run Keyword*. `repeat` specifies how many times (as a count) or how long time (as a timeout) the keyword should be executed.

If `repeat` is given as count, it specifies how many times the keyword should be executed. `repeat` can be given as an integer or as a string that can be converted to an integer. If it is a string, it can have postfix `times` or `x` (case and space insensitive) to make the expression more explicit.

If `repeat` is given as timeout, it must be in Robot Framework's time format (e.g. `1 minute`, `2 min 3 s`). Using a number alone (e.g. `1` or `1.5`) does not work in this context.

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

Specifying `repeat` as a timeout is new in Robot Framework 3.0.

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.

The file `template.txt` contains `Hello ${NAME}!` and variable `${NAME}` has the value `Robot`.

return_from_keyword(**return_values*)
Returns from the enclosing user keyword.

This keyword can be used to return from a user keyword with PASS status without executing it fully. It is also possible to return values similarly as with the `[Return]` setting. For more detailed information about working with the return values, see the User Guide.

This keyword is typically wrapped to some other keyword, such as *Run Keyword If* or *Run Keyword If Test Passed*, to return based on a condition:

It is possible to use this keyword to return from a keyword also inside a for loop. That, as well as returning values, is demonstrated by the *Find Index* keyword in the following somewhat advanced example. Notice that it is often a good idea to move this kind of complicated logic into a test library.

The most common use case, returning based on an expression, can be accomplished directly with *Return From Keyword If*. See also *Run Keyword And Return* and *Run Keyword And Return If*.

return_from_keyword_if(*condition*, **return_values*)
Returns from the enclosing user keyword if `condition` is true.

A wrapper for *Return From Keyword* to return based on the given condition. The condition is evaluated using the same semantics as with *Should Be True* keyword.

Given the same example as in *Return From Keyword*, we can rewrite the *Find Index* keyword as follows:

See also *Run Keyword And Return* and *Run Keyword And Return If*.

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*.

The execution is not continued if the failure is caused by invalid syntax, timeout, or fatal exception. Since Robot Framework 2.9, variable errors are caught by this keyword.

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

The keyword to execute and its arguments are specified using `name` and `*args` exactly like with *Run Keyword*.

The expected error must be given in the same format as in Robot Framework reports. By default it is interpreted as a glob pattern with `*`, `?` and `[chars]` as wildcards, but starting from Robot Framework 3.1 that can be changed by using various prefixes explained in the table below. Prefixes are case-sensitive and they must be separated from the actual message with a colon and an optional space like `PREFIX: Message` or `PREFIX:Message`.

See the *Pattern matching* section for more information about glob patterns and regular expressions.

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

Errors caused by invalid syntax, timeouts, or fatal exceptions are not caught by this keyword. Since Robot Framework 2.9, variable errors are 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 string `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.

Errors caused by invalid syntax, timeouts, or fatal exceptions are not caught by this keyword. Otherwise this keyword itself never fails. Since Robot Framework 2.9, variable errors are caught by this keyword.

run_keyword_and_return(*name*, **args*)
Runs the specified keyword and returns from the enclosing user keyword.

The keyword to execute is defined with `name` and `*args` exactly like with *Run Keyword*. After running the keyword, returns from the enclosing user keyword and passes possible return value from the executed keyword further. Returning from a keyword has exactly same semantics as with *Return From Keyword*.

Use *Run Keyword And Return If* if you want to run keyword and return based on a condition.

run_keyword_and_return_if(*condition*, *name*, **args*)
Runs the specified keyword and returns from the enclosing user keyword.

A wrapper for *Run Keyword And Return* to run and return based on the given `condition`. The condition is evaluated using the same semantics as with *Should Be True* keyword.

Use *Return From Keyword If* if you want to return a certain value based on a condition.

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 Boolean `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*.

Errors caused by invalid syntax, timeouts, or fatal exceptions are not caught by this keyword. Otherwise this keyword itself never fails.

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

The given `condition` is evaluated in Python as explained in *Evaluating expressions*, and `name` and `*args` have same semantics as with *Run Keyword*.

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*.

Variables used like `${variable}`, as in the examples above, are replaced in the expression before evaluation. Variables are also available in the evaluation namespace and can be accessed using special syntax `$variable`. This is a new feature in Robot Framework 2.9 and it is explained more thoroughly in *Evaluating expressions*.

This keyword supports also optional ELSE and ELSE IF branches. Both of them 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. Nested *Run Keyword If* usage is not supported when using ELSE and/or ELSE IF branches.

Given previous example, if/else construct can also be created like this:

The return value of this keyword is the return value of the actually executed keyword or Python `None` if no keyword was executed (i.e. if `condition` was false). Hence, it is recommended to use ELSE and/or ELSE IF branches to conditionally assign return values from keyword to variables (see *Set Variable If* if you need to set fixed values conditionally). This is illustrated by the example below:

In this example, ${var2} will be set to `None` if ${condition} is false.

Notice that `ELSE` and `ELSE IF` control words 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 escape them with a backslash like `\ELSE` and `\ELSE IF`.

Python's [http://docs.python.org/library/os.htmlos] and [http://docs.python.org/library/sys.htmlsys] modules are automatically imported when evaluating the `condition`. Attributes they contain can thus be used in the condition:

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.

Prior to Robot Framework 2.9 failures in test teardown itself were not detected by this keyword.

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.

Prior to Robot Framework 2.9 failures in test teardown itself were not detected by this keyword.

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.

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. Notice that this keyword does not support `ELSE` or `ELSE IF` branches like *Run Keyword If* does, though.

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.

By default all arguments are expected to be keywords to be executed.

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.

Notice that the `AND` control argument must be used explicitly and 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 subsequent test suites, test cases and user keywords. Also variables in variable tables are overridden. Variables assigned locally based on keyword return values or by using *Set Test Variable* and *Set Suite Variable* override these variables in that scope, but the global value is not changed in those cases.

In practice setting variables with this keyword has the same effect as using command line options `--variable` and `--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(**search_order*)
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

you can have

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:

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.
- 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, ERROR 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 a true value (see *Boolean arguments*), 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}`.

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 a true value (see *Boolean arguments*), 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.

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.

Possible child test suites do not see variables set with this keyword by default. Starting from Robot Framework 2.9, that can be controlled by using `children=<option>` as the last argument. If the specified `<option>` is a non-empty string or any other value considered true in Python, the variable is set also to the child suites. Parent and sibling suites will never 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`. Variable value can be given using the same syntax as when variables are created in the Variable table.

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.

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

NOTE: If the variable has value which itself is a variable (escaped or not), you must always use the escaped format to set the variable:

This limitation applies also to *Set Test Variable*, *Set Global Variable*, *Variable Should Exist*, *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_task_variable (*name*, **values*)
Makes a variable available everywhere within the scope of the current task.

This is an alias for *Set Test Variable* that is more applicable when creating tasks, not tests. New in RF 3.1.

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.

set_test_message (*message*, *append=False*)
Sets message for the current test case.

If the optional `append` argument is given a true value (see *Boolean arguments*), the given `message` is added after the possible earlier message by joining the messages with a space.

In test teardown this keyword can alter the possible failure message, but otherwise failures override messages set by this keyword. Notice that in teardown the message is available as a built-in variable `${TEST MESSAGE}`.

It is possible to use HTML format in the message by starting the message with `*HTML*`.

This keyword can not be used in suite setup or suite teardown.

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.

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.

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.

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*, *ignore_case=False*, *formatter='str'*)
Fails if the given objects are unequal.

Optional `msg`, `values` and `formatter` arguments specify how to construct the error message if this keyword fails:

- If `msg` is not given, the error message is `<first> != <second>`.
- If `msg` is given and `values` gets a true value (default), the error message is `<msg>: <first> != <second>`.
- If `msg` is given and `values` gets a false value (see *Boolean arguments*), the error message is simply `<msg>`.
- `formatter` controls how to format the values. Possible values are `str` (default), `repr` and `ascii`, and they work similarly as Python built-in functions with same names. See *String representations* for more details.

If `ignore_case` is given a true value (see *Boolean arguments*) and both arguments are strings, comparison is done case-insensitively. If both arguments are multiline strings, this keyword uses *multiline string comparison*.

`ignore_case` and `formatter` are new features in Robot Framework 3.0.1 and 3.1.2, respectively.

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`.

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`.

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 http://randomascii.wordpress.com/2012/02/25/comparing-floating-point-numbers-2012-edition/.

If you want to avoid possible problems with floating point numbers, you can implement custom keywords using Python's [http://docs.python.org/library/decimal.html|decimal] or [http://docs.python.org/library/fractions.html|fractions] modules.

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*, *ignore_case=False*, *formatter='str'*)
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`, `values` and `formatter`.

If `ignore_case` is given a true value (see *Boolean arguments*), comparison is done case-insensitively. If both arguments are multiline strings, this keyword uses *multiline string comparison*.

Strings are always [http://www.macchiato.com/unicode/nfc-faq| NFC normalized].

`ignore_case` and `formatter` are new features in Robot Framework 3.0.1 and 3.1.2, respectively.

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 as explained in *Evaluating expressions* and the keyword status is decided based on the result. If a non-string item is given, the status is got directly from its [http://docs.python.org/library/stdtypes.html#truth|truth value].

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

Variables used like `${variable}`, as in the examples above, are replaced in the expression before evaluation. Variables are also available in the evaluation namespace and can be accessed using special syntax `$variable`. This is a new feature in Robot Framework 2.9 and it is explained more thoroughly in *Evaluating expressions*.

Should Be True automatically imports Python's [http://docs.python.org/library/os.html|os] and [http://docs.python.org/library/sys.html|sys] modules that contain several useful attributes:

should_contain (*container*, *item*, *msg=None*, *values=True*, *ignore_case=False*)
Fails if `container` does not contain `item` one or more times.

Works with strings, lists, and anything that supports Python's `in` operator.

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

If `ignore_case` is given a true value (see *Boolean arguments*) and compared items are strings, it indicates that comparison should be case-insensitive. If the `container` is a list-like object, string items in it are compared case-insensitively. New option in Robot Framework 3.0.1.

should_contain_any(*container*, **items*, ***configuration*)
Fails if `container` does not contain any of the `*items`.

Works with strings, lists, and anything that supports Python's `in` operator.

Supports additional configuration parameters `msg`, `values` and `ignore_case`, which have exactly the same semantics as arguments with same names have with *Should Contain*. These arguments must always be given using `name=value` syntax after all `items`.

Note that possible equal signs in `items` must be escaped with a backslash (e.g. `foo\=bar`) to avoid them to be passed in as `**configuration`.

New in Robot Framework 3.0.1.

should_contain_x_times(*item1*, *item2*, *count*, *msg=None*, *ignore_case=False*)
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.

If `ignore_case` is given a true value (see *Boolean arguments*) and compared items are strings, it indicates that comparison should be case-insensitive. If the `item1` is a list-like object, string items in it are compared case-insensitively. New option in Robot Framework 3.0.1.

should_end_with(*str1*, *str2*, *msg=None*, *values=True*, *ignore_case=False*)
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`, as well as for semantics of the `ignore_case` option.

should_match(*string*, *pattern*, *msg=None*, *values=True*, *ignore_case=False*)
Fails if the given `string` does not match the given `pattern`.

Pattern matching is similar as matching files in a shell with `*`, `?` and `[chars]` acting as wildcards. See the *Glob patterns* section for more information.

See *Should Be Equal* for an explanation on how to override the default error message with `msg` and `values`, as well as for semantics of the `ignore_case` option.

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

See the *Regular expressions* section for more information about regular expressions and how to use then in Robot Framework test data.

Notice that 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`.

Possible flags altering how the expression is parsed (e.g. `re.IGNORECASE`, `re.MULTILINE`) must be embedded to the pattern like `(?im)pattern`. The most useful flags are `i` (case-insensitive), `m` (multiline mode), `s` (dotall mode) and `x` (verbose).

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.

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*, *ignore_case=False*)
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`.

If `ignore_case` is given a true value (see *Boolean arguments*) and both arguments are strings, comparison is done case-insensitively. New option in Robot Framework 3.0.1.

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`.

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*, *ignore_case=False*)
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`.

If `ignore_case` is given a true value (see *Boolean arguments*), comparison is done case-insensitively.

Strings are always [http://www.macchiato.com/unicode/nfc-faql NFC normalized].

`ignore_case` is a new feature in Robot Framework 3.0.1.

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 (*container*, *item*, *msg=None*, *values=True*, *ignore_case=False*)
Fails if `container` contains `item` one or more times.

Works with strings, lists, and anything that supports Python's `in` operator.

See *Should Be Equal* for an explanation on how to override the default error message with arguments `msg` and `values`. `ignore_case` has exactly the same semantics as with *Should Contain*.

should_not_contain_any (*container*, **items*, ***configuration*)
Fails if `container` contains one or more of the `*items`.

Works with strings, lists, and anything that supports Python's `in` operator.

Supports additional configuration parameters `msg`, `values` and `ignore_case`, which have exactly the same semantics as arguments with same names have with *Should Contain*. These arguments must always be given using `name=value` syntax after all `items`.

Note that possible equal signs in `items` must be escaped with a backslash (e.g. `foo\=bar`) to avoid them to be passed in as `**configuration`.

New in Robot Framework 3.0.1.

should_not_end_with (*str1*, *str2*, *msg=None*, *values=True*, *ignore_case=False*)
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`, as well as for semantics of the `ignore_case` option.

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

Pattern matching is similar as matching files in a shell with `*`, `?` and `[chars]` acting as wildcards. See the *Glob patterns* section for more information.

See *Should Be Equal* for an explanation on how to override the default error message with `msg` and `values`, as well as for semantics of the `ignore_case` option.

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*, *ignore_case=False*)
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`, as well as for semantics of the `ignore_case` option.

should_start_with (*str1*, *str2*, *msg=None*, *values=True*, *ignore_case=False*)
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`, as well as for semantics of the `ignore_case` option.

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.

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(*retry*, *retry_interval*, *name*, **args*)
Runs the specified keyword and retries if it fails.

`name` and `args` define the keyword that is executed similarly as with *Run Keyword*. How long to retry running the keyword is defined using `retry` argument either as timeout or count. `retry_interval` is the time to wait before trying to run the keyword again after the previous run has failed.

If `retry` is given as timeout, it must be in Robot Framework's time format (e.g. `1 minute`, `2 min 3 s`, `4.5`) that is explained in an appendix of Robot Framework User Guide. If it is given as count, it must have `times` or `x` postfix (e.g. `5 times`, `10 x`). `retry_interval` must always be given in Robot Framework's time format.

If the keyword does not succeed regardless of retries, this keyword fails. If the executed keyword passes, its return value is returned.

All normal failures are caught by this keyword. Errors caused by invalid syntax, test or keyword timeouts, or fatal exceptions (caused e.g. by *Fatal Error*) are not caught.

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

Support for specifying `retry` as a number of times to retry is a new feature in Robot Framework 2.9. Since Robot Framework 2.9, variable errors are caught by this keyword.

exception robot.libraries.BuiltIn.**RobotNotRunningError**
Bases: exceptions.AttributeError

Used when something cannot be done because Robot is not running.

Based on AttributeError to be backwards compatible with RF < 2.8.5. May later be based directly on Exception, so new code should except this exception explicitly.

args

message

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

NOTE: This API will change in RF 3.1. For more information see https://github.com/robotframework/robotframework/issues/2190. Use with *deprecation_warning=False* to avoid related deprecation warnings.

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).

Keywords executed by registered run keywords can be tested in dry-run mode if 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)

robot.libraries.Collections module

class robot.libraries.Collections.**NotSet**
Bases: object

class robot.libraries.Collections.**Collections**
Bases: robot.libraries.Collections._List, robot.libraries.Collections._Dictionary

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*).

= Related keywords in BuiltIn =

Following keywords in the BuiltIn library can also be used with lists and dictionaries:

= Using with list-like and dictionary-like objects =

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 Python list objects.

Similarly dictionary keywords can, for most parts, be used with other mappings. *Convert To Dictionary* can be used if real Python dict objects are needed.

= Boolean arguments =

Some keywords accept arguments that are handled as Boolean values true or false. If such an argument is given as a string, it is considered false if it is an empty string or equal to FALSE, NONE, NO, OFF or 0, case-insensitively. Keywords verifying something that allow dropping actual and expected values from the possible error message also consider string no values to be false. Other strings are considered true

regardless their value, and other argument types are tested using the same [http://docs.python.org/library/stdtypes.html#truth|rules as in Python].

True examples:

False examples:

Considering string `NONE` false is new in Robot Framework 3.0.3 and considering also `OFF` and `0` false is new in Robot Framework 3.1.

= Data in examples =

List related keywords use variables in format `${Lx}` in their examples. They mean lists with as many alphabetic characters as specified by `x`. For example, `${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 = '3.1.2'

should_contain_match(*list*, *pattern*, *msg=None*, *case_insensitive=False*, *whitespace_insensitive=False*)

Fails if `pattern` is not found in `list`.

By default, pattern matching is similar to matching files in a shell and is case-sensitive and whitespace-sensitive. In the pattern syntax, `*` matches to anything and `?` matches to any single character. You can also prepend `glob=` to your pattern to explicitly use this pattern matching behavior.

If you prepend `regexp=` to your pattern, your pattern will be used according to the Python [http://docs.python.org/library/re.html|re module] regular expression syntax. Important note: Backslashes are an escape character, and must be escaped with another backslash (e.g. `regexp=\\d{6}` to search for `\d{6}`). See *BuiltIn.Should Match Regexp* for more details.

If `case_insensitive` is given a true value (see *Boolean arguments*), the pattern matching will ignore case.

If `whitespace_insensitive` is given a true value (see *Boolean arguments*), the pattern matching will ignore whitespace.

Non-string values in lists are ignored when matching patterns.

Use the `msg` argument to override the default error message.

See also `Should Not Contain Match`.

should_not_contain_match(*list*, *pattern*, *msg=None*, *case_insensitive=False*, *whitespace_insensitive=False*)

Fails if `pattern` is found in `list`.

Exact opposite of *Should Contain Match* keyword. See that keyword for information about arguments and usage in general.

get_matches(*list*, *pattern*, *case_insensitive=False*, *whitespace_insensitive=False*)

Returns a list of matches to `pattern` in `list`.

For more information on `pattern`, `case_insensitive`, and `whitespace_insensitive`, see *Should Contain Match*.

get_match_count(*list*, *pattern*, *case_insensitive=False*, *whitespace_insensitive=False*)

Returns the count of matches to `pattern` in `list`.

For more information on `pattern`, `case_insensitive`, and `whitespace_insensitive`, see *Should Contain Match*.

append_to_list(*list_*, **values*)
Adds `values` to the end of `list`.

combine_lists(**lists*)
Combines the given `lists` together and returns the result.

The given lists are not altered by this keyword.

convert_to_dictionary(*item*)
Converts the given `item` to a Python `dict` type.

Mainly useful for converting other mappings to normal dictionaries. This includes converting Robot Framework's own `DotDict` instances that it uses if variables are created using the `&{var}` syntax.

Use *Create Dictionary* from the BuiltIn library for constructing new dictionaries.

New in Robot Framework 2.9.

convert_to_list(*item*)
Converts the given `item` to a Python `list` type.

Mainly useful for converting tuples and other iterable to lists. Use *Create List* from the BuiltIn library for constructing new lists.

copy_dictionary(*dictionary*, *deepcopy=False*)
Returns a copy of the given dictionary.

The `deepcopy` argument controls should the returned dictionary be a [https://docs.python.org/library/copy.html|shallow or deep copy]. By default returns a shallow copy, but that can be changed by giving `deepcopy` a true value (see *Boolean arguments*). This is a new option in Robot Framework 3.1.2. Earlier versions always returned shallow copies.

The given dictionary is never altered by this keyword.

copy_list(*list_*, *deepcopy=False*)
Returns a copy of the given list.

If the optional `deepcopy` is given a true value, the returned list is a deep copy. New option in Robot Framework 3.1.2.

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 with *Get Slice From List* keyword. 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. The types of the dictionaries do not need to be same.

See *Lists Should Be Equal* for more information about configuring the error message with `msg` and `values` arguments.

dictionary_should_contain_item(*dictionary*, *key*, *value*, *msg=None*)
An item of `key` / `value` must be found in a `dictionary`.

Value is converted to unicode for comparison.

Use the `msg` argument to override the default error message.

dictionary_should_contain_key (*dictionary*, *key*, *msg=None*)
Fails if `key` is not found from `dictionary`.

Use the `msg` argument to override the default error message.

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 more information about configuring the error message with `msg` and `values` arguments.

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

Use the `msg` argument to override the default error message.

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

Use the `msg` argument to override the default error message.

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

Use the `msg` argument to override the default error message.

get_dictionary_items (*dictionary*, *sort_keys=True*)
Returns items of the given `dictionary` as a list.

Uses *Get Dictionary Keys* to get keys and then returns corresponding items. By default keys are sorted and items returned in that order, but this can be changed by giving `sort_keys` a false value (see *Boolean arguments*). Notice that with Python 3.5 and earlier dictionary order is undefined unless using ordered dictionaries.

Items are returned as a flat list so that first item is a key, second item is a corresponding value, third item is the second key, and so on.

The given `dictionary` is never altered by this keyword.

`sort_keys` is a new option in Robot Framework 3.1.2. Earlier items were always sorted based on keys.

get_dictionary_keys (*dictionary*, *sort_keys=True*)
Returns keys of the given `dictionary` as a list.

By default keys are returned in sorted order (assuming they are sortable), but they can be returned in the original order by giving `sort_keys` a false value (see *Boolean arguments*). Notice that with Python 3.5 and earlier dictionary order is undefined unless using ordered dictionaries.

The given `dictionary` is never altered by this keyword.

`sort_keys` is a new option in Robot Framework 3.1.2. Earlier keys were always sorted.

get_dictionary_values (*dictionary*, *sort_keys=True*)
Returns values of the given `dictionary` as a list.

Uses *Get Dictionary Keys* to get keys and then returns corresponding values. By default keys are sorted and values returned in that order, but this can be changed by giving `sort_keys` a false value (see *Boolean arguments*). Notice that with Python 3.5 and earlier dictionary order is undefined unless using ordered dictionaries.

The given `dictionary` is never altered by this keyword.

`sort_keys` is a new option in Robot Framework 3.1.2. Earlier values were always sorted based on keys.

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.

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.

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 with *Get Slice From List* keyword. In case the value is not found, -1 is returned. The given list is never altered by this keyword.

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 and including the end, use `None` (default) as the end value.

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

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.

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.

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 *Lists Should Be Equal* for more information about configuring the error message with `msg` and `values` arguments.

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

Use the `msg` argument to override the default error message.

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 found from `list`.

Use the `msg` argument to override the default error message.

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 types of the lists do not need to be the same. For example, Python tuple and list with same content are considered equal.

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` gets a value considered true

(see *Boolean arguments*), the error message starts with the given `msg` followed by a newline and the default message.

- If `msg` is given and `values` is not given a true value, the error message is just the given `msg`.

Optional `names` argument 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.

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.

pop_from_dictionary (*dictionary*, *key*, *default=*)
Pops the given `key` from the `dictionary` and returns its value.

By default the keyword fails if the given `key` cannot be found from the `dictionary`. If optional `default` value is given, it will be returned instead of failing.

New in Robot Framework 2.9.2.

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.

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.

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.

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.

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.

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.

set_to_dictionary (*dictionary*, **key_value_pairs*, ***items*)
Adds the given `key_value_pairs` and `items` to the `dictionary`.

Giving items as `key_value_pairs` means giving keys and values as separate arguments:

The latter syntax is typically more convenient to use, but it has a limitation that keys must be strings.

If given keys already exist in the dictionary, their values are updated.

sort_list (*list_*)
Sorts the given list in place.

Sorting fails if items in the list are not comparable with each others. On Python 2 most objects are comparable, but on Python 3 comparing, for example, strings with numbers is not possible.

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

robot.libraries.DateTime module

A test library for handling date and time values.

`DateTime` is a Robot Framework standard library that supports creating and converting date and time values (e.g. *Get Current Date*, *Convert Time*), as well as doing simple calculations with them (e.g. *Subtract Time From Date*, *Add Time To Time*). It supports dates and times in various formats, and can also be used by other libraries programmatically.

= Table of Contents =

- *Terminology*

- *Date formats*
- *Time formats*
- *Millisecond handling*
- *Programmatic usage*
- *Shortcuts*
- *Keywords*

= Terminology =

In the context of this library, `date` and `time` generally have following meanings:

- **`date`: An entity with both date and time components but without any** timezone information. For example, `2014-06-11 10:07:42`.
- `time`: A time interval. For example, `1 hour 20 minutes` or `01:20:00`.

This terminology differs from what Python's standard [http://docs.python.org/library/datetime.html|datetime] module uses. Basically its [http://docs.python.org/library/datetime.html#datetime-objects|datetime] and [http://docs.python.org/library/datetime.html#timedelta-objects|timedelta] objects match `date` and `time` as defined by this library.

= Date formats =

Dates can given to and received from keywords in *timestamp*, *custom timestamp*, *Python datetime* and *epoch time* formats. These formats are discussed thoroughly in subsequent sections.

Input format is determined automatically based on the given date except when using custom timestamps, in which case it needs to be given using `date_format` argument. Default result format is timestamp, but it can be overridden using `result_format` argument.

== Timestamp ==

If a date is given as a string, it is always considered to be a timestamp. If no custom formatting is given using `date_format` argument, the timestamp is expected to be in [http://en.wikipedia.org/wiki/ISO_8601|ISO 8601] like format `YYYY-MM-DD hh:mm:ss.mil`, where any non-digit character can be used as a separator or separators can be omitted altogether. Additionally, only the date part is mandatory, all possibly missing time components are considered to be zeros.

Dates can also be returned in the same `YYYY-MM-DD hh:mm:ss.mil` format by using `timestamp` value with `result_format` argument. This is also the default format that keywords returning dates use. Milliseconds can be excluded using `exclude_millis` as explained in *Millisecond handling* section.

== Custom timestamp ==

It is possible to use custom timestamps in both input and output. The custom format is same as accepted by Python's [http://docs.python.org/library/datetime.html#strftime-strptime-behavior| datatime.strptime] function. For example, the default timestamp discussed in the previous section would match `%Y-%m-%d %H:%M:%S.%f`.

When using a custom timestamp in input, it must be specified using `date_format` argument. The actual input value must be a string that matches the specified format exactly. When using a custom timestamp in output, it must be given using `result_format` argument.

Notice that locale aware directives like `%b` do not work correctly with Jython on non-English locales: http://bugs.jython.org/issue2285

== Python datetime ==

Python's standard [http://docs.python.org/library/datetime.html#datetime-objects|datetime] objects can be used both in input and output. In input they are recognized automatically, and in output it is possible to get them by giving `datetime` value to `result_format` argument.

One nice benefit with datetime objects is that they have different time components available as attributes that can be easily accessed using the extended variable syntax.

== Epoch time ==

Epoch time is the time in seconds since the [http://en.wikipedia.org/wiki/Unix_time|UNIX epoch] i.e. 00:00:00.000 (UTC) 1 January 1970. To give a date in epoch time, it must be given as a number (integer or float), not as a string. To return a date in epoch time, it is possible to use `epoch` value with `result_format` argument. Epoch time is returned as a floating point number.

Notice that epoch time itself is independent on timezones and thus same around the world at a certain time. What local time a certain epoch time matches obviously then depends on the timezone. For example, examples below were tested in Finland but verifications would fail on other timezones.

== Earliest supported date ==

The earliest date that is supported depends on the date format and to some extend on the platform:

- Timestamps support year 1900 and above.
- Python datetime objects support year 1 and above.
- Epoch time supports 1970 and above on Windows with Python and IronPython.
- On other platforms epoch time supports 1900 and above or even earlier.

Prior to Robot Framework 2.9.2, all formats had same limitation as epoch time has nowadays.

= Time formats =

Similarly as dates, times can be given to and received from keywords in various different formats. Supported formats are *number*, *time string* (verbose and compact), *timer string* and *Python timedelta*.

Input format for time is always determined automatically based on the input. Result format is number by default, but it can be customised using `result_format` argument.

== Number ==

Time given as a number is interpreted to be seconds. It can be given either as an integer or a float, or it can be a string that can be converted to a number.

To return a time as a number, `result_format` argument must have value `number`, which is also the default. Returned number is always a float.

== Time string ==

Time strings are strings in format like `1 minute 42 seconds` or `1min 42s`. The basic idea of this format is having first a number and then a text specifying what time that number represents. Numbers can be either integers or floating point numbers, the whole format is case and space insensitive, and it is possible to add a minus prefix to specify negative times. The available time specifiers are:

- `days`, `day`, `d`
- `hours`, `hour`, `h`
- `minutes`, `minute`, `mins`, `min`, `m`
- `seconds`, `second`, `secs`, `sec`, `s`
- `milliseconds`, `millisecond`, `millis`, `ms`

When returning a time string, it is possible to select between `verbose` and `compact` representations using `result_format` argument. The verbose format uses long specifiers `day`, `hour`, `minute`, `second` and `millisecond`, and adds `s` at the end when needed. The compact format uses shorter specifiers `d`, `h`, `min`, `s` and `ms`, and even drops the space between the number and the specifier.

== Timer string ==

Timer string is a string given in timer like format `hh:mm:ss.mil`. In this format both hour and millisecond parts are optional, leading and trailing zeros can be left out when they are not meaningful, and negative times can be represented by adding a minus prefix.

To return a time as timer string, `result_format` argument must be given value `timer`. Timer strings are by default returned in full `hh:mm:ss.mil` format, but milliseconds can be excluded using `exclude_millis` as explained in *Millisecond handling* section.

== Python timedelta ==

Python's standard [http://docs.python.org/library/datetime.html#datetime.timedelta|timedelta] objects are also supported both in input and in output. In input they are recognized automatically, and in output it is possible to receive them by giving `timedelta` value to `result_format` argument.

= Millisecond handling =

This library handles dates and times internally using the precision of the given input. With *timestamp*, *time string*, and *timer string* result formats seconds are, however, rounded to millisecond accuracy. Milliseconds may also be included even if there would be none.

All keywords returning dates or times have an option to leave milliseconds out by giving a true value to `exclude_millis` argument. If the argument is given as a string, it is considered true unless it is empty or case-insensitively equal to `false`, `none` or `no`. Other argument types are tested using same [http://docs.python.org/library/stdtypes.html#truth|rules as in Python]. Notice that prior to Robot Framework 2.9, all strings except the empty string were considered true, and that considering `none` false is new in Robot Framework 3.0.3.

When milliseconds are excluded, seconds in returned dates and times are rounded to the nearest full second. With *timestamp* and *timer string* result formats, milliseconds will also be removed from the returned string altogether.

= Programmatic usage =

In addition to be used as normal library, this library is intended to provide a stable API for other libraries to use if they want to support same date and time formats as this library. All the provided keywords are available as functions that can be easily imported:

Additionally helper classes `Date` and `Time` can be used directly:

`robot.libraries.DateTime.`**`get_current_date`**(*time_zone='local'*, *increment=0*, *result_format='timestamp'*, *exclude_millis=False*)

Returns current local or UTC time with an optional increment.

Arguments: - `time_zone`: Get the current time on this time zone. Currently only `local` (default) and `UTC` are supported.

- **`increment`: Optional time increment to add to the returned date in** one of the supported *time formats*. Can be negative.
- `result_format`: Format of the returned date (see *date formats*).

- **exclude_millis: When set to any true value, rounds and drops** milliseconds as explained in *millisecond handling*.

robot.libraries.DateTime.**convert_date**(*date*, *result_format='timestamp'*, *exclude_millis=False*, *date_format=None*)

Converts between supported *date formats*.

Arguments: - date: Date in one of the supported *date formats*. - result_format: Format of the returned date. - exclude_millis: When set to any true value, rounds and drops

milliseconds as explained in *millisecond handling*.

- date_format: Specifies possible *custom timestamp* format.

robot.libraries.DateTime.**convert_time**(*time*, *result_format='number'*, *exclude_millis=False*)

Converts between supported *time formats*.

Arguments: - time: Time in one of the supported *time formats*. - result_format: Format of the returned time. - exclude_millis: When set to any true value, rounds and drops

milliseconds as explained in *millisecond handling*.

robot.libraries.DateTime.**subtract_date_from_date**(*date1*, *date2*, *result_format='number'*, *exclude_millis=False*, *date1_format=None*, *date2_format=None*)

Subtracts date from another date and returns time between.

Arguments: - date1: Date to subtract another date from in one of the

supported *date formats*.

- **date2: Date that is subtracted in one of the supported** *date formats*.
- result_format: Format of the returned time (see *time formats*).
- **exclude_millis: When set to any true value, rounds and drops** milliseconds as explained in *millisecond handling*.
- date1_format: Possible *custom timestamp* format of date1.
- date2_format: Possible *custom timestamp* format of date2.

Examples:

robot.libraries.DateTime.**add_time_to_date**(*date*, *time*, *result_format='timestamp'*, *exclude_millis=False*, *date_format=None*)

Adds time to date and returns the resulting date.

Arguments: - date: Date to add time to in one of the supported

date formats.

- **time: Time that is added in one of the supported** *time formats*.
- result_format: Format of the returned date.
- **exclude_millis: When set to any true value, rounds and drops** milliseconds as explained in *millisecond handling*.
- date_format: Possible *custom timestamp* format of date.

robot.libraries.DateTime.**subtract_time_from_date**(*date*, *time*, *result_format='timestamp'*, *exclude_millis=False*, *date_format=None*)

Subtracts time from date and returns the resulting date.

Arguments: - `date`: Date to subtract time from in one of the supported

date formats.

- **`time`: Time that is subtracted in one of the supported** *time formats*.
- `result_format`: Format of the returned date.
- **`exclude_millis`: When set to any true value, rounds and drops** milliseconds as explained in *millisecond handling*.
- `date_format`: Possible *custom timestamp* format of `date`.

robot.libraries.DateTime.**add_time_to_time**(*time1*, *time2*, *result_format='number'*, *exclude_millis=False*)

Adds time to another time and returns the resulting time.

Arguments: - `time1`: First time in one of the supported *time formats*. - `time2`: Second time in one of the supported *time formats*. - `result_format`: Format of the returned time. - `exclude_millis`: When set to any true value, rounds and drops

milliseconds as explained in *millisecond handling*.

robot.libraries.DateTime.**subtract_time_from_time**(*time1*, *time2*, *result_format='number'*, *exclude_millis=False*)

Subtracts time from another time and returns the resulting time.

Arguments: - `time1`: Time to subtract another time from in one of

the supported *time formats*.

- `time2`: Time to subtract in one of the supported *time formats*.
- `result_format`: Format of the returned time.
- **`exclude_millis`: When set to any true value, rounds and drops** milliseconds as explained in *millisecond handling*.

robot.libraries.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 whether tests are run on Python, IronPython or Jython but they provide the same functionality.

Long lines in the provided messages are wrapped automatically. If you want to wrap lines manually, you can add newlines using the `\n` character sequence.

The library has a known limitation that it cannot be used with timeouts on Python. Support for IronPython was added in Robot Framework 2.9.2.

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 press either PASS or FAIL button. In the latter case execution fails and 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=''*, *hidden=False*)
Pauses test execution and asks user to input a value.

Value typed by the user, or the possible default value, is returned. Returning an empty value is fine, but pressing Cancel fails the keyword.

message is the instruction shown in the dialog and default_value is the possible default value shown in the input field.

If hidden is given a true value, the value typed by the user is hidden. hidden is considered true if it is a non-empty string not equal to false, none or no, case-insensitively. If it is not a string, its truth value is got directly using same [http://docs.python.org/library/stdtypes.html#truth|rules as in Python].

Considering strings false and no to be false is new in RF 2.9 and considering string none false is new in RF 3.0.3.

robot.libraries.Dialogs.**get_selection_from_user**(*message*, **values*)
Pauses test execution and asks user to select a value.

The selected value is returned. Pressing Cancel fails the keyword.

message is the instruction shown in the dialog and values are the options given to the user.

robot.libraries.Dialogs.**get_selections_from_user**(*message*, **values*)
Pauses test execution and asks user to select multiple values.

The selected values are returned as a list. Selecting no values is OK and in that case the returned list is empty. Pressing Cancel fails the keyword.

message is the instruction shown in the dialog and values are the options given to the user.

New in Robot Framework 3.1.

robot.libraries.Easter module

robot.libraries.Easter.**none_shall_pass**(*who*)

robot.libraries.OperatingSystem module

class robot.libraries.OperatingSystem.**OperatingSystem**
Bases: object

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*).

== Table of contents ==

- *Path separators*
- *Pattern matching*
- *Tilde expansion*
- *Boolean arguments*
- *Example*
- *Shortcuts*
- *Keywords*

= Path separators =

Because Robot Framework uses the backslash (\) as an escape character in the test data, using a literal backslash requires duplicating it like in `c:\\path\\file.txt`. That can be inconvenient especially with longer Windows paths, and thus all keywords expecting paths as arguments convert forward slashes to backslashes automatically on Windows. This also means that paths like `${CURDIR}/path/file.txt` are operating system independent.

Notice that the automatic path separator conversion does not work if the path is only a part of an argument like with *Run* and *Start Process* keywords. In these cases the built-in variable `${/}` that contains `\` or `/`, depending on the operating system, can be used instead.

= Pattern matching =

Some keywords allow their arguments to be specified as [http://en.wikipedia.org/wiki/Glob_(programming)|glob patterns] where:

Unless otherwise noted, matching is case-insensitive on case-insensitive operating systems such as Windows.

Starting from Robot Framework 2.9.1, globbing is not done if the given path matches an existing file even if it would contain a glob pattern.

= Tilde expansion =

Paths beginning with `~` or `~username` are expanded to the current or specified user's home directory, respectively. The resulting path is operating system dependent, but typically e.g. `~/robot` is expanded to `C:\Users\<user>\robot` on Windows and `/home/<user>/robot` on Unixes.

The `~username` form does not work on Jython.

= Boolean arguments =

Some keywords accept arguments that are handled as Boolean values true or false. If such an argument is given as a string, it is considered false if it is an empty string or equal to `FALSE`, `NONE`, `NO`, `OFF` or `0`, case-insensitively. Other strings are considered true regardless their value, and other argument types are tested using the same [http://docs.python.org/library/stdtypes.html#truth|rules as in Python].

True examples:

False examples:

Considering string `NONE` false is new in Robot Framework 3.0.3 and considering also `OFF` and `0` false is new in Robot Framework 3.1.

= Example =

```
ROBOT_LIBRARY_SCOPE = 'GLOBAL'

ROBOT_LIBRARY_VERSION = '3.1.2'
```

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.

TIP: Run Process keyword provided by the [http://robotframework.org/robotframework/latest/libraries/Process.html| Process library] supports better process configuration and is generally recommended as a replacement for this keyword.

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).

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

TIP: Run Process keyword provided by the [http://robotframework.org/robotframework/latest/libraries/Process.html| Process library] supports better process configuration and is generally recommended as a replacement for this keyword.

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*.

TIP: Run Process keyword provided by the [http://robotframework.org/robotframework/latest/libraries/Process.html| Process library] supports better process configuration and is generally recommended as a replacement for this keyword.

get_file(*path*, *encoding='UTF-8'*, *encoding_errors='strict'*)
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. The default value is `UTF-8`, which means that UTF-8 and ASCII encoded files are read correctly. In addition to the encodings supported by the underlying Python implementation, the following special encoding values can be used:

- `SYSTEM`: Use the default system encoding.
- `CONSOLE`: Use the console encoding. Outside Windows this is same as the system encoding.

`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.

Support for `SYSTEM` and `CONSOLE` encodings in Robot Framework 3.0.

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*.

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

This keyword reads a file from the file system using the defined `path`, `encoding` and `encoding_errors` 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.

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'*, *encoding_errors='strict'*)
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.

See *Get File* for more information about `encoding` and `encoding_errors` arguments.

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 in an appendix of Robot Framework User Guide.

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 in an appendix of Robot Framework User Guide.

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 the file is created does not exist, it is automatically created along with possible missing intermediate directories. Possible existing file is overwritten.

On Windows newline characters (`\n`) in content are automatically converted to Windows native newline sequence (`\r\n`).

See *Get File* for more information about possible `encoding` values, including special values `SYSTEM` and `CONSOLE`.

Use *Append To File* if you want to append to an existing file and *Create Binary File* if you need to write bytes without encoding. *File Should Not Exist* can be used to avoid overwriting existing files.

The support for `SYSTEM` and `CONSOLE` encodings is new in Robot Framework 3.0. Automatically converting `\n` to `\r\n` on Windows is new in Robot Framework 3.1.

create_binary_file(*path*, *content*)
Creates a binary file with the given content.

If content is given as a Unicode string, it is first converted to bytes character by character. All characters with ordinal below 256 can be used and are converted to bytes with same values. Using characters with higher ordinal is an error.

Byte strings, and possible other types, are written to the file as is.

If the directory for the file does not exist, it is created, along with missing intermediate directories.

Use *Create File* if you want to create a text file using a certain encoding. *File Should Not Exist* can be used to avoid overwriting existing files.

append_to_file(*path*, *content*, *encoding='UTF-8'*)
Appends the given content to the specified file.

If the file exists, the given text is written to its end. If the file does not exist, it is created.

Other than not overwriting possible existing files, this keyword works exactly like *Create File*. See its documentation for more details about the usage.

Note that special encodings `SYSTEM` and `CONSOLE` only work with this keyword starting from Robot Framework 3.1.2.

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.

empty_directory(*path*)
Deletes all the content from the given directory.

Deletes both files and sub-directories, but the specified directory itself if not removed. Use *Remove Directory* if you want to remove the whole directory.

create_directory(*path*)
Creates the specified directory.

Also possible intermediate directories are created. Passes if the directory already exists, but fails if the path exists and is not a directory.

remove_directory(*path*, *recursive=False*)
Removes the directory pointed to by the given `path`.

If the second argument `recursive` is given a true value (see *Boolean arguments*), 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 the destination.

Source must be a path to an existing file or a glob pattern (see *Pattern matching*) that matches exactly one file. How the destination is interpreted is explained below.

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 as the source 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.

The resulting destination path is returned since Robot Framework 2.9.2.

See also *Copy Files*, *Move File*, and *Move Files*.

move_file(*source*, *destination*)
Moves the source file into the destination.

Arguments have exactly same semantics as with *Copy File* keyword. Destination file path is returned since Robot Framework 2.9.2.

If the source and destination are on the same filesystem, rename operation is used. Otherwise file is copied to the destination filesystem and then removed from the original filesystem.

See also *Move Files*, *Copy File*, and *Copy Files*.

copy_files(**sources_and_destination*)
Copies specified files to the target directory.

Source files can be given as exact paths and as glob patterns (see *Pattern matching*). At least one source must be given, but it is not an error if it is a pattern that does not match anything.

Last argument must be the destination directory. If the destination does not exist, it will be created.

See also *Copy File*, *Move File*, and *Move Files*.

move_files(**sources_and_destination*)
Moves specified files to the target directory.

Arguments have exactly same semantics as with *Copy Files* keyword.

See also *Move File*, *Copy File*, and *Copy Files*.

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.

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. Set variables are automatically encoded using the system encoding.

append_to_environment_variable(*name*, **values*, ***config*)
Appends given `values` to environment variable `name`.

If the environment variable already exists, values are added after it, and otherwise a new environment variable is created.

Values are, by default, joined together using the operating system path separator (`;` on Windows, `:` elsewhere). This can be changed by giving a separator after the values like `separator=value`. No other configuration parameters are accepted.

remove_environment_variable(**names*)
Deletes the specified environment variable.

Does nothing if the environment variable is not set.

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.

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.

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.

- ${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.

- @{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*, *case_normalize=False*)
Normalizes the given path.

- Collapses redundant separators and up-level references.
- Converts / to \ on Windows.
- Replaces initial ~ or `~user` by that user's home directory. The latter is not supported on Jython.
- If `case_normalize` is given a true value (see *Boolean arguments*) on Windows, converts the path to all lowercase. New in Robot Framework 3.1.
- ${path1} = 'abc'
- ${path2} = 'def'
- ${path3} = 'abc/def/ghi'
- ${path4} = '/home/robot/stuff'

On Windows result would use \ instead of / and home directory would be different.

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.

- ${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.

- ${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`.

2006-03-29 15:06:21): - ${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. Modified time can only be set to regular files.

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.
4) If `mtime` is equal to `UTC`, the current time in [http://en.wikipedia.org/wiki/Coordinated_Universal_Time|UTC] is used.
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.

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.

File and directory names are returned relative to the given path (e.g. `'file.txt'`) by default. If you want them be returned in absolute format (e.g. `'/home/robot/file.txt'`), give the `absolute` argument a true value (see *Boolean arguments*).

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.

list_files_in_directory (*path*, *pattern=None*, *absolute=False*)
Wrapper for *List Directory* that returns only files.

list_directories_in_directory (*path*, *pattern=None*, *absolute=False*)
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 with *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*)
Wrapper for *Count Items In Directory* returning only file count.

count_directories_in_directory(*path*, *pattern=None*)
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.

robot.libraries.Process module

class robot.libraries.Process.**Process**
Bases: `object`

Robot Framework test library for running processes.

This library utilizes Python's [http://docs.python.org/library/subprocess.html|subprocess] module and its [http://docs.python.org/library/subprocess.html#popen-constructor|Popen] class.

The library has following main usages:

- Running processes in system and waiting for their completion using *Run Process* keyword.
- Starting processes on background using *Start Process*.
- Waiting started process to complete using *Wait For Process* or stopping them with *Terminate Process* or *Terminate All Processes*.

== Table of contents ==

- *Specifying command and arguments*
- *Process configuration*
- *Active process*
- *Result object*
- *Boolean arguments*
- *Example*
- *Shortcuts*
- *Keywords*

= Specifying command and arguments =

Both *Run Process* and *Start Process* accept the command to execute and all arguments passed to the command as separate arguments. This makes usage convenient and also allows these keywords to automatically escape possible spaces and other special characters in commands and arguments. Notice that if a command accepts options that themselves accept values, these options and their values must be given as separate arguments.

When *running processes in shell*, it is also possible to give the whole command to execute as a single string. The command can then contain multiple commands to be run together. When using this approach, the caller is responsible on escaping.

Possible non-string arguments are converted to strings automatically.

= Process configuration =

Run Process and *Start Process* keywords can be configured using optional `**configuration` keyword arguments. Configuration arguments must be given after other arguments passed to these keywords and must use syntax like `name=value`. Available configuration arguments are listed below and discussed further in sections afterwards.

Note that because `**configuration` is passed using `name=value` syntax, possible equal signs in other arguments passed to *Run Process* and *Start Process* must be escaped with a backslash like `name\=value`. See *Run Process* for an example.

== Running processes in shell ==

The `shell` argument specifies whether to run the process in a shell or not. By default shell is not used, which means that shell specific commands, like `copy` and `dir` on Windows, are not available. You can, however, run shell scripts and batch files without using a shell.

Giving the `shell` argument any non-false value, such as `shell=True`, changes the program to be executed in a shell. It allows using the shell capabilities, but can also make the process invocation operating system dependent. Having a shell between the actually started process and this library can also interfere communication with the process such as stopping it and reading its outputs. Because of these problems, it is recommended to use the shell only when absolutely necessary.

When using a shell it is possible to give the whole command to execute as a single string. See *Specifying command and arguments* section for examples and more details in general.

== Current working directory ==

By default the child process will be executed in the same directory as the parent process, the process running tests, is executed. This can be changed by giving an alternative location using the `cwd` argument. Forward slashes in the given path are automatically converted to backslashes on Windows.

Standard output and error streams, when redirected to files, are also relative to the current working directory possibly set using the `cwd` argument.

== Environment variables ==

By default the child process will get a copy of the parent process's environment variables. The `env` argument can be used to give the child a custom environment as a Python dictionary. If there is a need to specify only certain environment variable, it is possible to use the `env:<name>=<value>` format to set or override only that named variables. It is also possible to use these two approaches together.

== Standard output and error streams ==

By default processes are run so that their standard output and standard error streams are kept in the memory. This works fine normally, but if there is a lot of output, the output buffers may get full and the program can hang. Additionally on Jython, everything written to these in-memory buffers can be lost if the process is terminated.

To avoid the above mentioned problems, it is possible to use `stdout` and `stderr` arguments to specify files on the file system where to redirect the outputs. This can also be useful if other processes or other keywords need to read or manipulate the outputs somehow.

Given `stdout` and `stderr` paths are relative to the *current working directory*. Forward slashes in the given paths are automatically converted to backslashes on Windows.

As a special feature, it is possible to redirect the standard error to the standard output by using `stderr=STDOUT`.

Regardless are outputs redirected to files or not, they are accessible through the *result object* returned when the process ends. Commands are expected to write outputs using the console encoding, but *output encoding* can be configured using the `output_encoding` argument if needed.

Note that the created output files are not automatically removed after the test run. The user is responsible to remove them if needed.

== Output encoding ==

Executed commands are, by default, expected to write outputs to the *standard output and error streams* using the encoding used by the system console. If the command uses some other encoding, that can be configured using the `output_encoding` argument. This is especially useful on Windows where the console uses a different encoding than rest of the system, and many commands use the general system encoding instead of the console encoding.

The value used with the `output_encoding` argument must be a valid encoding and must match the encoding actually used by the command. As a convenience, it is possible to use strings `CONSOLE` and `SYSTEM` to specify that the console or system encoding is used, respectively. If produced outputs use different encoding then configured, values got through the *result object* will be invalid.

The support to set output encoding is new in Robot Framework 3.0.

== Alias ==

A custom name given to the process that can be used when selecting the *active process*.

= Active process =

The test library keeps record which of the started processes is currently active. By default it is latest process started with *Start Process*, but *Switch Process* can be used to select a different one. Using *Run Process* does not affect the active process.

The keywords that operate on started processes will use the active process by default, but it is possible to explicitly select a different process using the `handle` argument. The handle can be the identifier returned by *Start Process* or an `alias` explicitly given to *Start Process* or *Run Process*.

= Result object =

Run Process, *Wait For Process* and *Terminate Process* keywords return a result object that contains information about the process execution as its attributes. The same result object, or some of its attributes, can also be get using *Get Process Result* keyword. Attributes available in the object are documented in the table below.

= Boolean arguments =

Some keywords accept arguments that are handled as Boolean values true or false. If such an argument is given as a string, it is considered false if it is an empty string or equal to `FALSE`, `NONE`, `NO`, `OFF` or `0`, case-insensitively. Other strings are considered true regardless their value, and other argument types are tested using the same [http://docs.python.org/library/stdtypes.html#truth|rules as in Python].

True examples:

False examples:

Considering string `NONE` false is new in Robot Framework 3.0.3 and considering also `OFF` and `0` false is new in Robot Framework 3.1.

= Example =

```
ROBOT_LIBRARY_SCOPE = 'GLOBAL'
```

ROBOT_LIBRARY_VERSION = '3.1.2'

TERMINATE_TIMEOUT = 30

KILL_TIMEOUT = 10

run_process(*command*, **arguments*, ***configuration*)
Runs a process and waits for it to complete.

`command` and `*arguments` specify the command to execute and arguments passed to it. See *Specifying command and arguments* for more details.

`**configuration` contains additional configuration related to starting processes and waiting for them to finish. See *Process configuration* for more details about configuration related to starting processes. Configuration related to waiting for processes consists of `timeout` and `on_timeout` arguments that have same semantics as with *Wait For Process* keyword. By default there is no timeout, and if timeout is defined the default action on timeout is `terminate`.

Returns a *result object* containing information about the execution.

Note that possible equal signs in `*arguments` must be escaped with a backslash (e.g. `name\=value`) to avoid them to be passed in as `**configuration`.

This keyword does not change the *active process*.

start_process(*command*, **arguments*, ***configuration*)
Starts a new process on background.

See *Specifying command and arguments* and *Process configuration* for more information about the arguments, and *Run Process* keyword for related examples.

Makes the started process new *active process*. Returns an identifier that can be used as a handle to activate the started process if needed.

Processes are started so that they create a new process group. This allows sending signals to and terminating also possible child processes. This is not supported on Jython.

is_process_running(*handle=None*)
Checks is the process running or not.

If `handle` is not given, uses the current *active process*.

Returns `True` if the process is still running and `False` otherwise.

process_should_be_running(*handle=None*, *error_message='Process is not running.'*)
Verifies that the process is running.

If `handle` is not given, uses the current *active process*.

Fails if the process has stopped.

process_should_be_stopped(*handle=None*, *error_message='Process is running.'*)
Verifies that the process is not running.

If `handle` is not given, uses the current *active process*.

Fails if the process is still running.

wait_for_process(*handle=None*, *timeout=None*, *on_timeout='continue'*)
Waits for the process to complete or to reach the given timeout.

The process to wait for must have been started earlier with *Start Process*. If `handle` is not given, uses the current *active process*.

`timeout` defines the maximum time to wait for the process. It can be given in [http://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#time-format| various time formats] supported by Robot Framework, for example, `42`, `42 s`, or `1 minute 30 seconds`.

`on_timeout` defines what to do if the timeout occurs. Possible values and corresponding actions are explained in the table below. Notice that reaching the timeout never fails the test.

See *Terminate Process* keyword for more details how processes are terminated and killed.

If the process ends before the timeout or it is terminated or killed, this keyword returns a *result object* containing information about the execution. If the process is left running, Python `None` is returned instead.

terminate_process(*handle=None*, *kill=False*)
Stops the process gracefully or forcefully.

If `handle` is not given, uses the current *active process*.

By default first tries to stop the process gracefully. If the process does not stop in 30 seconds, or `kill` argument is given a true value, (see *Boolean arguments*) kills the process forcefully. Stops also all the child processes of the originally started process.

Waits for the process to stop after terminating it. Returns a *result object* containing information about the execution similarly as *Wait For Process*.

On Unix-like machines graceful termination is done using `TERM (15)` signal and killing using `KILL (9)`. Use *Send Signal To Process* instead if you just want to send either of these signals without waiting for the process to stop.

On Windows graceful termination is done using `CTRL_BREAK_EVENT` event and killing using Win32 API function `TerminateProcess()`.

Limitations: - Graceful termination is not supported on Windows when using Jython.

Process is killed instead.

- Stopping the whole process group is not supported when using Jython.
- On Windows forceful kill only stops the main process, not possible child processes.

terminate_all_processes(*kill=False*)
Terminates all still running processes started by this library.

This keyword can be used in suite teardown or elsewhere to make sure that all processes are stopped,

By default tries to terminate processes gracefully, but can be configured to forcefully kill them immediately. See *Terminate Process* that this keyword uses internally for more details.

send_signal_to_process(*signal*, *handle=None*, *group=False*)
Sends the given `signal` to the specified process.

If `handle` is not given, uses the current *active process*.

Signal can be specified either as an integer as a signal name. In the latter case it is possible to give the name both with or without `SIG` prefix, but names are case-sensitive. For example, all the examples below send signal `INT (2)`:

This keyword is only supported on Unix-like machines, not on Windows. What signals are supported depends on the system. For a list of existing signals on your system, see the Unix man pages related to signal handling (typically `man signal` or `man 7 signal`).

By default sends the signal only to the parent process, not to possible child processes started by it. Notice that when *running processes in shell*, the shell is the parent process and it depends on the system does the shell propagate the signal to the actual started process.

To send the signal to the whole process group, `group` argument can be set to any true value (see *Boolean arguments*). This is not supported by Jython, however.

get_process_id(*handle=None*)
Returns the process ID (pid) of the process as an integer.

If `handle` is not given, uses the current *active process*.

Notice that the pid is not the same as the handle returned by *Start Process* that is used internally by this library.

get_process_object(*handle=None*)
Return the underlying `subprocess.Popen` object.

If `handle` is not given, uses the current *active process*.

get_process_result(*handle=None*, *rc=False*, *stdout=False*, *stderr=False*, *stdout_path=False*, *stderr_path=False*)
Returns the specified *result object* or some of its attributes.

The given `handle` specifies the process whose results should be returned. If no `handle` is given, results of the current *active process* are returned. In either case, the process must have been finishes before this keyword can be used. In practice this means that processes started with *Start Process* must be finished either with *Wait For Process* or *Terminate Process* before using this keyword.

If no other arguments than the optional `handle` are given, a whole *result object* is returned. If one or more of the other arguments are given any true value, only the specified attributes of the *result object* are returned. These attributes are always returned in the same order as arguments are specified in the keyword signature. See *Boolean arguments* section for more details about true and false values.

Although getting results of a previously executed process can be handy in general, the main use case for this keyword is returning results over the remote library interface. The remote interface does not support returning the whole result object, but individual attributes can be returned without problems.

switch_process(*handle*)
Makes the specified process the current *active process*.

The handle can be an identifier returned by *Start Process* or the `alias` given to it explicitly.

split_command_line(*args*, *escaping=False*)
Splits command line string into a list of arguments.

String is split from spaces, but argument surrounded in quotes may contain spaces in them. If `escaping` is given a true value, then backslash is treated as an escape character. It can escape unquoted spaces, quotes inside quotes, and so on, but it also requires using double backslashes when using Windows paths.

New in Robot Framework 2.9.2.

join_command_line(**args*)
Joins arguments into one command line string.

In resulting command line string arguments are delimited with a space, arguments containing spaces are surrounded with quotes, and possible quotes are escaped with a backslash.

If this keyword is given only one argument and that is a list like object, then the values of that list are joined instead.

New in Robot Framework 2.9.2.

class robot.libraries.Process.**ExecutionResult**(*process*, *stdout*, *stderr*, *rc=None*, *output_encoding=None*)

Bases: `object`

stdout

stderr

close_streams()

class robot.libraries.Process.**ProcessConfiguration**(*cwd=None*, *shell=False*, *stdout=None*, *stderr=None*, *output_encoding='CONSOLE'*, *alias=None*, *env=None*, ***rest*)

Bases: object

get_command(*command*, *arguments*)

popen_config

result_config

robot.libraries.Remote module

class robot.libraries.Remote.**Remote**(*uri='http://127.0.0.1:8270'*, *timeout=None*)

Bases: object

Connects to a remote server at `uri`.

Optional `timeout` can be used to specify a timeout to wait when initially connecting to the server and if a connection accidentally closes. Timeout can be given as seconds (e.g. `60`) or using Robot Framework time format (e.g. `60s`, `2 minutes 10 seconds`).

The default timeout is typically several minutes, but it depends on the operating system and its configuration. Notice that setting a timeout that is shorter than keyword execution time will interrupt the keyword.

Timeouts do not work with IronPython.

ROBOT_LIBRARY_SCOPE = 'TEST SUITE'

get_keyword_names(*attempts=2*)

get_keyword_arguments(*name*)

get_keyword_types(*name*)

get_keyword_tags(*name*)

get_keyword_documentation(*name*)

run_keyword(*name*, *args*, *kwargs*)

class robot.libraries.Remote.**ArgumentCoercer**

Bases: object

binary = <_sre.SRE_Pattern object>

non_ascii = <_sre.SRE_Pattern object>

coerce(*argument*)

class robot.libraries.Remote.**RemoteResult**(*result*)

Bases: object

class robot.libraries.Remote.**XmlRpcRemoteClient**(*uri*, *timeout=None*)

Bases: object

get_keyword_names()

get_keyword_arguments(*name*)

get_keyword_types(*name*)

get_keyword_tags(*name*)

get_keyword_documentation(*name*)

run_keyword(*name*, *args*, *kwargs*)

class robot.libraries.Remote.**TimeoutHTTPTransport**(*use_datetime=0*, *timeout=None*)
Bases: xmlrpclib.Transport

make_connection(*host*)

accept_gzip_encoding = True

close()

encode_threshold = None

get_host_info(*host*)

getparser()

parse_response(*response*)

request(*host*, *handler*, *request_body*, *verbose=0*)

send_content(*connection*, *request_body*)

send_host(*connection*, *host*)

send_request(*connection*, *handler*, *request_body*)

send_user_agent(*connection*)

single_request(*host*, *handler*, *request_body*, *verbose=0*)

user_agent = 'xmlrpclib.py/1.0.1 (by www.pythonware.com)'

class robot.libraries.Remote.**TimeoutHTTPSTransport**(*use_datetime=0*, *timeout=None*)
Bases: *robot.libraries.Remote.TimeoutHTTPTransport*

accept_gzip_encoding = True

close()

encode_threshold = None

get_host_info(*host*)

getparser()

make_connection(*host*)

parse_response(*response*)

request(*host*, *handler*, *request_body*, *verbose=0*)

send_content(*connection*, *request_body*)

send_host(*connection*, *host*)

send_request(*connection*, *handler*, *request_body*)

send_user_agent(*connection*)

single_request(*host*, *handler*, *request_body*, *verbose=0*)

user_agent = 'xmlrpclib.py/1.0.1 (by www.pythonware.com)'

robot.libraries.Reserved module

class robot.libraries.Reserved.**Reserved**

Bases: object

ROBOT_LIBRARY_SCOPE = 'GLOBAL'

get_keyword_names()

run_keyword(*name*, *args*)

robot.libraries.Screenshot module

class robot.libraries.Screenshot.**Screenshot**(*screenshot_directory=None*, *screenshot_module=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 =

How screenshots are taken when using Python depends on the operating system. On OSX screenshots are taken using the built-in `screencapture` utility. On other operating systems you need to have one of the following tools or Python modules installed. You can specify the tool/module to use when *importing* the library. If no tool or module is specified, the first one 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.
- Pillow :: http://python-pillow.github.io :: Only works on Windows. Also the original PIL package is supported.
- Scrot :: http://en.wikipedia.org/wiki/Scrot :: Not used on Windows. Install with `apt-get install scrot` or similar.

Using `screencapture` on OSX and specifying explicit screenshot module are new in Robot Framework 2.9.2. The support for using `scrot` is new in Robot Framework 3.0.

= 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.

= 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 when *importing* the library and using *Set Screenshot Directory* keyword during execution. It is also possible to save screenshots using an absolute path.

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.

`screenshot_module` specifies the module or tool to use when using this library on Python outside OSX. Possible values are `wxPython`, `PyGTK`, `PIL` and `scrot`, case-insensitively. If no value is given, the first module/tool found is used in that order. See *Using with Python* for more information.

Specifying explicit screenshot module is new in Robot Framework 2.9.2.

ROBOT_LIBRARY_SCOPE = 'TEST SUITE'

ROBOT_LIBRARY_VERSION = '3.1.2'

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*.

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.

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`

test(*path=None*)

robot.libraries.String module

class robot.libraries.String.**String**
Bases: `object`

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*
- *Convert To Bytes*

ROBOT_LIBRARY_SCOPE = 'GLOBAL'

ROBOT_LIBRARY_VERSION = '3.1.2'

convert_to_lowercase(*string*)
Converts string to lowercase.

convert_to_uppercase(*string*)
Converts string to uppercase.

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

Use *Convert To Bytes* in `BuiltIn` if you want to create bytes based on character or integer sequences. 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.

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

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.

format_string(*template*, **positional*, ***named*)
Formats a `template` using the given `positional` and `named` arguments.

The template can be either be a string or an absolute path to an existing file. In the latter case the file is read and its contents are used as the template. If the template file contains non-ASCII characters, it must be encoded using UTF-8.

The template is formatted using Python's [https://docs.python.org/library/string.html#format-string-syntax|format string syntax]. Placeholders are marked using `{}` with possible field name and format specification inside. Literal curly braces can be inserted by doubling them like *{{* and *}}*.

New in Robot Framework 3.1.

get_line_count(*string*)
Returns and logs the number of lines in the given string.

split_to_lines (*string*, *start=0*, *end=None*)
Splits the given string to 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.

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.

Use *Split To Lines* if all lines are needed.

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, not a glob or regexp pattern. A line matches if the `pattern` is found anywhere on it.

The match is case-sensitive by default, but giving `case_insensitive` a true value makes it case-insensitive. The value is considered true if it is a non-empty string that is not equal to `false`, `none` or `no`. If the value is not a string, its truth value is got directly in Python. Considering `none` false is new in RF 3.0.3.

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.

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:

A line matches only if it matches the `pattern` fully.

The match is case-sensitive by default, but giving `case_insensitive` a true value makes it case-insensitive. The value is considered true if it is a non-empty string that is not equal to `false`, `none` or `no`. If the value is not a string, its truth value is got directly in Python. Considering `none` false is new in RF 3.0.3.

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.

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*, *partial_match=False*)
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.

By default lines match only if they match the pattern fully, but partial matching can be enabled by giving the `partial_match` argument a true value. The value is considered true if it is a non-empty string that is not equal to `false`, `none` or `no`. If the value is not a string, its truth value is got directly in Python. Considering `none` false is new in RF 3.0.3.

If the pattern is empty, it matches only empty lines by default. When partial matching is enabled, empty pattern matches all lines.

Notice that to make the match case-insensitive, you need to prefix the pattern with case-insensitive flag `(?i)`.

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.

See *Get Lines Matching Pattern* and *Get Lines Containing String* if you do not need full regular expression powers (and complexity).

`partial_match` argument is new in Robot Framework 2.9. In earlier versions exact match was always required.

`get_regexp_matches`(*string*, *pattern*, **groups*)
Returns a list of all non-overlapping matches in the given string.

`string` is the string to find matches from and `pattern` is the 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.

If no groups are used, the returned list contains full matches. If one group is used, the list contains only contents of that group. If multiple groups are used, the list contains tuples that contain individual group contents. All groups can be given as indexes (starting from 1) and named groups also as names.

New in Robot Framework 2.9.

`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 you need to just remove a string see *Remove String*.

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.

`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.

If you need to just remove a string see *Remove String Using Regexp*.

`remove_string`(*string*, **removables*)
Removes all `removables` from the given `string`.

`removables` are used as literal strings. Each removable will be matched to a temporary string from which preceding removables have been already removed. See second example below.

Use *Remove String Using Regexp* if more powerful pattern matching is needed. If only a certain number of matches should be removed, *Replace String* or *Replace String Using Regexp* can be used.

A modified version of the string is returned and the original string is not altered.

`remove_string_using_regexp`(*string*, **patterns*)
Removes `patterns` from the given `string`.

This keyword is otherwise identical to *Remove String*, but the `patterns` to search for are considered to be a regular expression. See *Replace String Using Regexp* for more information about the regular expression syntax. That keyword can also be used if there is a need to remove only a certain number of occurrences.

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.

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.

split_string_to_characters(*string*)
Splits the given `string` to characters.

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:

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.

strip_string(*string*, *mode='both'*, *characters=None*)
Remove leading and/or trailing whitespaces from the given string.

`mode` is either `left` to remove leading characters, `right` to remove trailing characters, `both` (default) to remove the characters from both sides of the string or `none` to return the unmodified string.

If the optional `characters` is given, it must be a string and the characters in the string will be stripped in the string. Please note, that this is not a substring to be removed but a list of characters, see the example below.

New in Robot Framework 3.0.

should_be_string(*item*, *msg=None*)
Fails if the given `item` is not a string.

With Python 2, except with IronPython, this keyword passes regardless is the `item` 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. Notice that with Python 2, except with IronPython, `'string'` creates a byte string and `u'unicode'` must be used to create a Unicode string.

With Python 3 and IronPython, this keyword passes if the string is a Unicode string but fails if it is bytes. Notice that with both Python 3 and IronPython, `'string'` creates a Unicode string, and `b'bytes'` must be used to create a byte string.

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.

See *Should Be String* for more details about Unicode strings and byte strings.

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. See *Should Be String* for more details about Unicode strings and byte strings.

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

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. See *Should Be String* for more details about Unicode strings and byte strings.

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

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*.

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*.

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*.

robot.libraries.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'*, *window_size=None*, *environ_user=None*, *terminal_emulation=False*, *terminal_type=None*, *telnetlib_log_level='TRACE'*, *connection_timeout=None*)

Bases: `object`

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*
- *Terminal emulation*
- *Logging*
- *Time string format*
- *Boolean arguments*
- *Importing*
- *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].

The ANSI escape codes, like cursor movement and color codes, are normally returned as part of the read operation. If an escape code occurs in middle of a search pattern it may also prevent finding the searched string. *Terminal emulation* can be used to process these escape codes as they would be if a real terminal would be in use.

= 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*, *Set Default Log Level* and *Set Telnetlib Log Level*.

Values of `environ_user`, `window_size`, `terminal_emulation`, and `terminal_type` can not be changed after opening the connection.

== 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.

== Connection Timeout ==

Connection Timeout defines how long is the maximum time to wait when opening the telnet connection. It is used internally by *Open Connection*. The default value is the system global default timeout.

New in Robot Framework 2.9.2.

== 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.

== 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. Prompt can be set to be a regular expression by giving `prompt_is_regexp` argument a true value (see *Boolean arguments*).

== 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

== 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.

== Terminal type ==

By default the Telnet library does not negotiate any specific terminal type with the server. If a specific terminal type, for example `vt100`, is desired, the terminal type can be configured in *importing* and with *Open Connection*.

== Window size ==

Window size for negotiation with the server can be configured when *importing* the library and with *Open Connection*.

== USER environment variable ==

Telnet protocol allows the `USER` environment variable to be sent when connecting to the server. On some servers it may happen that there is no login prompt, and on those cases this configuration option will allow still to define the desired username. The option `environ_user` can be used in *importing* and with *Open Connection*.

= Terminal emulation =

Telnet library supports terminal emulation with [http://pyte.readthedocs.io|Pyte]. Terminal emulation will process the output in a virtual screen. This means that ANSI escape codes, like cursor movements, and also control characters, like carriage returns and backspaces, have the same effect on the result as they would have on a normal terminal screen. For example the sequence `acdc\x1b[3Dbba` will result in output `abba`.

Terminal emulation is taken into use by giving `terminal_emulation` argument a true value (see *Boolean arguments*) either in the library initialization or with *Open Connection*.

As Pyte approximates vt-style terminal, you may also want to set the terminal type as `vt100`. We also recommend that you increase the window size, as the terminal emulation will break all lines that are longer than the window row length.

When terminal emulation is used, the *newline* and *encoding* can not be changed anymore after opening the connection.

As a prerequisite for using terminal emulation, you need to have Pyte installed. Due to backwards incompatible changes in Pyte, different Robot Framework versions support different Pyte versions:

- Pyte 0.6 and newer are supported by Robot Framework 3.0.3. Latest Pyte version can be installed (or upgraded) with `pip install --upgrade pyte`.
- Pyte 0.5.2 and older are supported by Robot Framework 3.0.2 and earlier. Pyte 0.5.2 can be installed with `pip install pyte==0.5.2`.

= 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/library/telnetlib.html|telnetlib module] used by this library has a custom logging system for logging content it sends and receives. By default these messages are written using `TRACE` level, but the level is configurable with the `telnetlib_log_level` option either in the library initialization, to the *Open Connection* or by using the *Set Telnetlib Log Level* keyword to the active connection. Special level `NONE` con be used to disable the logging altogether.

= 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://robotframework.org/robotframework/#user-guide|Robot Framework User Guide].

= Boolean arguments =

Some keywords accept arguments that are handled as Boolean values true or false. If such an argument is given as a string, it is considered false if it is an empty string or equal to `FALSE`, `NONE`, `NO`, `OFF` or `0`, case-insensitively. Other strings are considered true regardless their value, and other argument types are tested using the same [http://docs.python.org/library/stdtypes.html#truth|rules as in Python].

True examples:

False examples:

Considering string `NONE` false is new in Robot Framework 3.0.3 and considering also `OFF` and `0` false is new in Robot Framework 3.1.

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 ... keywords*. See these keywords as well as *Configuration*, *Terminal emulation* and *Logging* sections above for more information about these parameters and their possible values.

See *Time string format* and *Boolean arguments* sections for information about using arguments accepting times and Boolean values, respectively.

ROBOT_LIBRARY_SCOPE = 'TEST_SUITE'

ROBOT_LIBRARY_VERSION = '3.1.2'

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*, *window_size=None*, *environ_user=None*, *terminal_emulation=None*, *terminal_type=None*, *telnetlib_log_level=None*, *connection_timeout=None*)

Opens a new Telnet connection to the given host and port.

The `timeout`, `newline`, `prompt`, `prompt_is_regexp`, `encoding`, `default_log_level`, `window_size`, `environ_user`, `terminal_emulation`, `terminal_type` and `telnetlib_log_level` arguments get default values when the library is [#Importing|imported]. Setting them here overrides those values for the opened connection. See *Configuration*, *Terminal emulation* and *Logging* sections for more information about these parameters and their possible values.

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.

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.

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 is 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'*, *window_size=None*, *environ_user=None*, *terminal_emulation=False*, *terminal_type=None*, *telnetlib_log_level='TRACE'*, *connection_timeout=None*)

Bases: telnetlib.Telnet

NEW_ENVIRON_IS = '\x00'

NEW_ENVIRON_VAR = '\x00'

NEW_ENVIRON_VALUE = '\x01'

INTERNAL_UPDATE_FREQUENCY = 0.03

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.

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.

If terminal emulation is used, the newline can not be changed on an open connection.

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 a true value (see *Boolean arguments*), the given prompt is considered to be a regular expression.

The old prompt is returned and can be used to restore the prompt later.

See the documentation of [http://docs.python.org/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.

If terminal emulation is used, the encoding can not be changed on an open connection.

set_telnetlib_log_level(*level*)
Sets the log level used for *logging* in the underlying `telnetlib`.

Note that `telnetlib` can be very noisy thus using the level `NONE` can shutdown the messages generated by this library.

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.

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.

The above example writes command `ps -ef | grep myprocess\r\n` 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.

write_control_character(*character*)
Writes the given control character into the connection.

The control character is prepended with an IAC (interpret as command) character.

The following control character names are supported: BRK, IP, AO, AYT, EC, EL, NOP. Additionally, you can use arbitrary numbers to send any control character.

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/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.

read_until_prompt(*loglevel=None*, *strip_prompt=False*)
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.

By default, 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].

If you want to exclude the prompt from the returned output, set `strip_prompt` to a true value (see *Boolean arguments*). If your prompt is a regular expression, make sure that the expression spans the whole prompt, because only the part of the output that matches the regular expression is stripped away.

See *Logging* section for more information about log levels.

execute_command(*command*, *loglevel=None*, *strip_prompt=False*)
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:

See *Logging* section for more information about log levels and *Read Until Prompt* for more information about the `strip_prompt` parameter.

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>*)
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.

class robot.libraries.Telnet.**TerminalEmulator**(*window_size=None*, *newline='rn'*)
Bases: object

current_output

feed(*text*)

read()

read_until(*expected*)

read_until_regexp(*regexp_list*)

exception robot.libraries.Telnet.**NoMatchError**(*expected*, *timeout*, *output=None*)
Bases: exceptions.AssertionError

ROBOT_SUPPRESS_NAME = True

args

message

robot.libraries.XML module

class robot.libraries.XML.**XML**(*use_lxml=False*)
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 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*
- *Using lxml*
- *Example*
- *Finding elements with xpath*
- *Element attributes*
- *Handling XML namespaces*
- *Boolean arguments*

- *Pattern matching*
- *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. Possible comments and processing instructions in the source XML are removed.

XML is not validated during parsing even if has a schema defined. How possible doctype elements are handled otherwise depends on the used XML module and on the platform. The standard ElementTree strips doctypes altogether but when *using lxml* they are preserved when XML is saved.

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*. Notice that keywords that modify XML do not write those changes back to disk even if the source would be given as a path to a file. Changes must always saved explicitly using *Save XML* keyword.

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.

= Using lxml =

By default this library uses Python's standard [http://docs.python.org/library/xml.etree.elementtree.html|ElementTree] module for parsing XML, but it can be configured to use [http://lxml.de|lxml] module instead when *importing* the library. The resulting element structure has same API regardless which module is used for parsing.

The main benefits of using lxml is that it supports richer xpath syntax than the standard ElementTree and enables using *Evaluate Xpath* keyword. It also preserves the doctype and possible namespace prefixes saving XML.

= 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.

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 standard. The supported xpath syntax is explained below and [https://docs.python.org/library/xml.etree.elementtree.html#xpath-support| ElementTree documentation] provides more details. In the examples `${XML}` refers to the same XML structure as in the earlier example.

If lxml support is enabled when *importing* the library, the whole [http://www.w3.org/TR/xpath/|xpath 1.0 standard] is supported. That includes everything listed below but also lot of other useful constructs.

== Tag names ==

When just a single tag name is used, xpath matches all direct child elements that have that tag name.

== 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.

== Wildcards ==

An asterisk (`*`) can be used in paths instead of a tag name to denote any element.

== 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.

== 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.

== 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 earlier. What predicates the standard ElementTree supports is explained in the table below.

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#element-objects|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.

== 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.

== 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.

== attrib ==

A Python dictionary containing attributes of the element.

= Handling XML namespaces =

ElementTree and lxml handle 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. Alternatively *Parse XML* supports stripping namespace information altogether by using `strip_namespaces` argument. The pros and cons of different 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:

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.

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:

Also this output is semantically same as the original. If the original XML had only default namespaces, the output would also look identical.

== Namespaces when using lxml ==

This library handles namespaces same way both when *using lxml* and when not using it. There are, however, differences how lxml internally handles namespaces compared to the standard ElementTree. The main difference is that lxml stores information about namespace prefixes and they are thus preserved if XML is saved. Another visible difference is that lxml includes namespace information in child elements got with *Get Element* if the parent element has namespaces.

== Stripping namespaces altogether ==

Because namespaces often add unnecessary complexity, *Parse XML* supports stripping them altogether by using `strip_namespaces=True`. When this option is enabled, namespaces are not shown anywhere nor are they included if XML is saved.

== 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.

= Boolean arguments =

Some keywords accept arguments that are handled as Boolean values true or false. If such an argument is given as a string, it is considered false if it is an empty string or equal to `FALSE`, `NONE`, `NO`, `OFF` or `0`, case-insensitively. Other strings are considered true regardless their value, and other argument types are tested using the same [http://docs.python.org/library/stdtypes.html#truth|rules as in Python].

True examples:

False examples:

Considering string `NONE` false is new in Robot Framework 3.0.3 and considering also `OFF` and `0` false is new in Robot Framework 3.1.

== Pattern matching ==

Some keywords, for example *Elements Should Match*, support so called [http://en.wikipedia.org/wiki/Glob_(programming)|glob patterns] where:

Unlike with glob patterns normally, path separator characters `/` and `\` and the newline character `\n` are matches by the above wildcards.

Support for brackets like `[abc]` and `[!a-z]` is new in Robot Framework 3.1

Import library with optionally lxml mode enabled.

By default this library uses Python's standard [http://docs.python.org/library/xml.etree.elementtree.html|ElementTree] module for parsing XML. If `use_lxml` argument is given a true value (see *Boolean arguments*), the library will use [http://lxml.de|lxml] module instead. See *Using lxml* section for benefits provided by lxml.

Using lxml requires that the lxml module is installed on the system. If lxml mode is enabled but the module is not installed, this library will emit a warning and revert back to using the standard ElementTree.

ROBOT_LIBRARY_SCOPE = 'GLOBAL'

ROBOT_LIBRARY_VERSION = '3.1.2'

parse_xml(*source*, *keep_clark_notation=False*, *strip_namespaces=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#element-objects|element structure] and the root element is returned. Possible comments and processing instructions in the source XML are removed.

As discussed in *Handling XML namespaces* section, this keyword, by default, removes namespace information ElementTree has added to tag names and moves it into `xmlns` attributes. 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 (see *Boolean arguments*).

If you want to strip namespace information altogether so that it is not included even if XML is saved, you can give a true value to `strip_namespaces` argument. This functionality is new in Robot Framework 3.0.2.

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.

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.

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.

Many other keywords use this keyword internally, and keywords modifying XML are typically documented to both to modify the given source and to return it. Modifying the source does not apply if the source is given as a string. The XML structure parsed based on the string and then modified is nevertheless returned.

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.

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.

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*.

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.

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.

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 contain. If the element has no text, an empty string is returned. The returned text is thus not always the same as the *text* attribute of the element.

By default all whitespace, including newlines and indentation, inside the element is returned as-is. If `normalize_whitespace` is given a true value (see *Boolean arguments*), 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.

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.

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.

element_text_should_match(*source*, *pattern*, *xpath='.'*, *normalize_whitespace=False*, *message=None*)
Verifies that the text of the specified element matches `expected`.

This keyword works exactly like *Element Text Should Be* except that the expected value can be given as a pattern that the text of the element must match.

Pattern matching is similar as matching files in a shell with `*`, `?` and `[chars]` acting as wildcards. See the *Pattern matching* section for more information.

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.

See also *Get Element Attributes*, *Element Attribute Should Be*, *Element Attribute Should Match* and *Element Should Not Have 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.

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*.

See also *Element Attribute Should Match* and *Get Element Attribute*.

element_attribute_should_match(*source*, *name*, *pattern*, *xpath='.'*, *message=None*)
Verifies that the specified attribute 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 with `*`, `?` and `[chars]` acting as wildcards. See the *Pattern matching* section for more information.

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.

See also *Get Element Attribute*, *Get Element Attributes*, *Element Text Should Be* and *Element Text Should Match*.

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 a true value (see *Boolean arguments*).

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 a 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*.

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. This includes the `.` at the end that is the *tail* text of the `<i>` element.

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 with `*`, `?` and `[chars]` acting as wildcards. See the *Pattern matching* section for more information.

See *Elements Should Be Equal* for more examples.

set_element_tag(*source*, *tag*, *xpath='.'*)
Sets the tag of the specified element.

The element whose tag to set is specified using `source` and `xpath`. They have exactly the same semantics as with *Get Element* keyword. The resulting XML structure is returned, and if the `source` is an already parsed XML structure, it is also modified in place.

Can only set the tag of a single element. Use *Set Elements Tag* to set the tag of multiple elements in one call.

set_elements_tag(*source*, *tag*, *xpath='.'*)
Sets the tag of the specified elements.

Like *Set Element Tag* but sets the tag of all elements matching the given `xpath`.

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 resulting XML structure is returned, and if the `source` is an already parsed XML structure, it is also modified in place.

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.

Can only set the text/tail of a single element. Use *Set Elements Text* to set the text/tail of multiple elements in one call.

set_elements_text(*source*, *text=None*, *tail=None*, *xpath='.'*)
Sets text and/or tail text of the specified elements.

Like *Set Element Text* but sets the text or tail of all elements matching the given `xpath`.

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 resulting XML structure is returned, and if the `source` is an already parsed XML structure, it is also modified in place.

It is possible to both set new attributes and to overwrite existing. Use *Remove Element Attribute* or *Remove Element Attributes* for removing them.

Can only set an attribute of a single element. Use *Set Elements Attribute* to set an attribute of multiple elements in one call.

set_elements_attribute(*source*, *name*, *value*, *xpath='.'*)
Sets attribute `name` of the specified elements to `value`.

Like *Set Element Attribute* but sets the attribute of all elements matching the given `xpath`.

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 resulting XML structure is returned, and if the `source` is an already parsed XML structure, it is also modified in place.

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.

Can only remove an attribute from a single element. Use *Remove Elements Attribute* to remove an attribute of multiple elements in one call.

remove_elements_attribute(*source*, *name*, *xpath='.'*)
Removes attribute `name` from the specified elements.

Like *Remove Element Attribute* but removes the attribute of all elements matching the given `xpath`.

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 resulting XML structure is returned, and if the `source` is an already parsed XML structure, it is also modified in place.

Use *Remove Element Attribute* to remove a single attribute and *Set Element Attribute* to set them.

Can only remove attributes from a single element. Use *Remove Elements Attributes* to remove all attributes of multiple elements in one call.

remove_elements_attributes(*source*, *xpath='.'*)
Removes all attributes from the specified elements.

Like *Remove Element Attributes* but removes all attributes of all elements matching the given `xpath`.

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 resulting XML structure is returned, and if the `source` is an already parsed XML structure, it is also modified in place.

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.).

Use *Remove Element* or *Remove Elements* to remove elements.

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 resulting XML structure is returned, and if the `source` is an already parsed XML structure, it is also modified in place.

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 (see *Boolean arguments*). See *Element attributes* section for more information about *tail* in general.

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 resulting XML structure is returned, and if the `source` is an already parsed XML structure, it is also modified in place.

It is not a failure if `xpath` matches no elements. Use *Remove Element* to remove exactly one element.

Element's tail text is not removed by default, but that can be changed by using `remove_tail` argument similarly as with *Remove Element*.

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 resulting XML structure is returned, and if the `source` is an already parsed XML structure, it is also modified in place.

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 (see *Boolean arguments*). See *Element attributes* section for more information about tail in general.

Use *Remove Element* to remove the whole element.

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.

element_to_string (*source*, *xpath='.'*, *encoding=None*)
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.

By default the string is returned as Unicode. If `encoding` argument is given any value, the string is returned as bytes in the specified encoding. The resulting string never contains the 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 always contains the XML declaration.

The resulting XML file may not be exactly the same as the original: - Comments and processing instructions are always stripped. - Possible doctype and namespace prefixes are only preserved when

using lxml.

- Other small differences are possible depending on the ElementTree or lxml version.

Use *Element To String* if you just need a string representation of the element.

evaluate_xpath(*source*, *expression*, *context='.'*)
Evaluates the given xpath expression and returns results.

The element in which context the expression is executed is specified using `source` and `context` arguments. They have exactly the same semantics as `source` and `xpath` arguments have with *Get Element* keyword.

The xpath expression to evaluate is given as `expression` argument. The result of the evaluation is returned as-is.

This keyword works only if lxml mode is taken into use when *importing* the library.

class robot.libraries.XML.**NameSpaceStripper**(*etree*, *lxml_etree=False*)
Bases: object

strip(*elem*, *preserve=True*, *current_ns=None*, *top=True*)

unstrip(*elem*, *current_ns=None*, *copied=False*)

class robot.libraries.XML.**ElementFinder**(*etree*, *modern=True*, *lxml=False*)
Bases: object

find_all(*elem*, *xpath*)

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*)

robot.libraries.dialogs_ipy module

robot.libraries.dialogs_jy module

robot.libraries.dialogs_py module

class robot.libraries.dialogs_py.**MessageDialog**(*message*, *value=None*, ***extra*)
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, 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 `__builtin__.float`

getint
alias of `__builtin__.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(*timeout=30*)

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 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.

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 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_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 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.

class robot.libraries.dialogs_py.**InputDialog**(*message*, *default=''*, *hidden=False*)
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, 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 `__builtin__.float`

getint
alias of `__builtin__.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(*timeout=30*)

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 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.

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 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_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 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.

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(*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 `__builtin__.float`

getint
alias of `__builtin__.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(*timeout=30*)

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 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.

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 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_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 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.

class robot.libraries.dialogs_py.**MultipleSelectionDialog**(*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(*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 `__builtin__.float`

getint
alias of `__builtin__.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(*timeout=30*)

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 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.

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 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_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 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.

class robot.libraries.dialogs_py.**PassFailDialog**(*message*, *value=None*, ***extra*)
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, 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 `__builtin__.float`

getint
alias of `__builtin__.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(*timeout=30*)

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 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.

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 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_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 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.

robot.model package

Package with generic, reusable and extensible model classes.

This package contains, for example, *TestSuite*, *TestCase*, *Keyword* and *SuiteVisitor* base classes. These classes are extended both by *execution* and *result* related model objects and used also elsewhere.

This package is considered stable.

Submodules

robot.model.configurer module

class robot.model.configurer.**SuiteConfigurer**(*name=None*, *doc=None*, *metadata=None*, *set_tags=None*, *include_tags=None*, *exclude_tags=None*, *include_suites=None*, *include_tests=None*, *empty_suite_ok=False*)

Bases: *robot.model.visitor.SuiteVisitor*

add_tags

remove_tags

visit_suite(*suite*)
Implements traversing through the suite and its direct children.

Can be overridden to allow modifying the passed in `suite` without calling *start_suite()* or *end_suite()* nor visiting child suites, tests or keywords (setup and teardown) at all.

end_keyword(*keyword*)
Called when keyword ends. Default implementation does nothing.

end_message(*msg*)
Called when message ends. Default implementation does nothing.

end_suite(*suite*)
Called when suite ends. Default implementation does nothing.

end_test(*test*)
Called when test ends. Default implementation does nothing.

start_keyword(*keyword*)
Called when keyword starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_message(*msg*)
Called when message starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_suite(*suite*)
Called when suite starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_test(*test*)
Called when test starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

visit_keyword(*kw*)
Implements traversing through the keyword and its child keywords.

Can be overridden to allow modifying the passed in `kw` without calling *start_keyword()* or *end_keyword()* nor visiting child keywords.

visit_message(*msg*)
Implements visiting the message.

Can be overridden to allow modifying the passed in `msg` without calling *start_message()* or *end_message()*.

visit_test(*test*)
Implements traversing through the test and its keywords.

Can be overridden to allow modifying the passed in `test` without calling *start_test()* or *end_test()* nor visiting keywords.

robot.model.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*)

robot.model.filter module

class robot.model.filter.**EmptySuiteRemover**
Bases: *robot.model.visitor.SuiteVisitor*

end_suite(*suite*)
Called when suite ends. Default implementation does nothing.

visit_test(*test*)
Implements traversing through the test and its keywords.

Can be overridden to allow modifying the passed in `test` without calling *start_test()* or *end_test()* nor visiting keywords.

visit_keyword(*kw*)
Implements traversing through the keyword and its child keywords.

Can be overridden to allow modifying the passed in `kw` without calling *start_keyword()* or *end_keyword()* nor visiting child keywords.

end_keyword(*keyword*)
Called when keyword ends. Default implementation does nothing.

end_message(*msg*)
Called when message ends. Default implementation does nothing.

end_test(*test*)
Called when test ends. Default implementation does nothing.

start_keyword(*keyword*)
Called when keyword starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_message(*msg*)
Called when message starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_suite(*suite*)
Called when suite starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_test(*test*)
Called when test starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

visit_message(*msg*)
Implements visiting the message.

Can be overridden to allow modifying the passed in `msg` without calling *start_message()* or *end_message()*.

visit_suite(*suite*)
Implements traversing through the suite and its direct children.

Can be overridden to allow modifying the passed in `suite` without calling *start_suite()* or *end_suite()* nor visiting child suites, tests or keywords (setup and teardown) at all.

class robot.model.filter.**Filter**(*include_suites=None*, *include_tests=None*, *include_tags=None*, *exclude_tags=None*)
Bases: *robot.model.filter.EmptySuiteRemover*

include_suites

include_tests

include_tags

exclude_tags

start_suite(*suite*)
Called when suite starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

end_keyword(*keyword*)
Called when keyword ends. Default implementation does nothing.

end_message(*msg*)
Called when message ends. Default implementation does nothing.

end_suite(*suite*)
Called when suite ends. Default implementation does nothing.

end_test(*test*)
Called when test ends. Default implementation does nothing.

start_keyword(*keyword*)
Called when keyword starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_message(*msg*)
Called when message starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_test(*test*)
Called when test starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

visit_keyword(*kw*)
Implements traversing through the keyword and its child keywords.

Can be overridden to allow modifying the passed in `kw` without calling *start_keyword()* or *end_keyword()* nor visiting child keywords.

visit_message(*msg*)
Implements visiting the message.

Can be overridden to allow modifying the passed in `msg` without calling *start_message()* or *end_message()*.

visit_suite(*suite*)
Implements traversing through the suite and its direct children.

Can be overridden to allow modifying the passed in `suite` without calling *start_suite()* or *end_suite()* nor visiting child suites, tests or keywords (setup and teardown) at all.

visit_test(*test*)
Implements traversing through the test and its keywords.

Can be overridden to allow modifying the passed in `test` without calling *start_test()* or *end_test()* nor visiting keywords.

robot.model.imports module

class robot.model.imports.**Import**(*type*, *name*, *args=()*, *alias=None*, *source=None*)
Bases: `object`

ALLOWED_TYPES = ('Library', 'Resource', 'Variables')

directory

report_invalid_syntax(*message*, *level='ERROR'*)

class robot.model.imports.**Imports**(*source*, *imports=None*)
Bases: *robot.model.itemlist.ItemList*

library(*name*, *args=()*, *alias=None*)

resource(*path*)

variables(*path*, *args=()*)

append(*item*)

clear()

create(**args*, ***kwargs*)

extend(*items*)

index(*item*, **start_and_end*)

insert(*index*, *item*)

pop(**index*)

visit(*visitor*)

robot.model.itemlist module

class robot.model.itemlist.**ItemList**(*item_class*, *common_attrs=None*, *items=None*)

Bases: object

create(**args*, ***kwargs*)

append(*item*)

extend(*items*)

insert(*index*, *item*)

pop(**index*)

index(*item*, **start_and_end*)

clear()

visit(*visitor*)

robot.model.keyword module

class robot.model.keyword.**Keyword**(*name=''*, *doc=''*, *args=()*, *assign=()*, *tags=()*, *timeout=None*, *type='kw'*)

Bases: *robot.model.modelobject.ModelObject*

Base model for a single keyword.

Extended by *robot.running.model.Keyword* and *robot.result.model.Keyword*.

KEYWORD_TYPE = 'kw'
Normal keyword *type*.

SETUP_TYPE = 'setup'
Setup *type*.

TEARDOWN_TYPE = 'teardown'
Teardown *type*.

FOR_LOOP_TYPE = 'for'
For loop *type*.

FOR_ITEM_TYPE = 'foritem'
Single for loop iteration *type*.

keyword_class = None
Internal usage only.

message_class
alias of *robot.model.message.Message*

doc

args
Keyword arguments as a list of strings.

assign
Assigned variables as a list of strings.

timeout

type
Keyword type as a string. The value is either *KEYWORD_TYPE*, *SETUP_TYPE*, *TEARDOWN_TYPE*, *FOR_LOOP_TYPE* or *FOR_ITEM_TYPE* constant defined on the class level.

name

parent
Parent test suite, test case or keyword.

tags
Keyword tags as a *Tags* object.

keywords
Child keywords as a *Keywords* object.

messages
Messages as a *Messages* object.

children
Child *keywords* and *messages* in creation order.

id
Keyword id in format like `s1-t3-k1`.

See *TestSuite.id* for more information.

visit(*visitor*)
Visitor interface entry-point.

copy(***attributes*)
Return shallow copy of this object.

Parameters **attributes** – Attributes to be set for the returned copy automatically. For example, `test.copy(name='New name')`.

See also *deepcopy()*. The difference between these two is the same as with the standard `copy.copy` and `copy.deepcopy` functions that these methods also use internally.

New in Robot Framework 3.0.1.

deepcopy(***attributes*)
Return deep copy of this object.

Parameters **attributes** – Attributes to be set for the returned copy automatically. For example, `test.deepcopy(name='New name')`.

See also *copy()*. The difference between these two is the same as with the standard `copy.copy` and `copy.deepcopy` functions that these methods also use internally.

New in Robot Framework 3.0.1.

class robot.model.keyword.**Keywords**(*keyword_class=<class 'robot.model.keyword.Keyword'>*, *parent=None*, *keywords=None*)
Bases: *robot.model.itemlist.ItemList*

A list-like object representing keywords in a suite, a test or a keyword.

Possible setup and teardown keywords are directly available as *setup* and *teardown* attributes.

append(*item*)

clear()

create(**args*, ***kwargs*)

extend(*items*)

index(*item*, **start_and_end*)

insert(*index*, *item*)

pop(**index*)

visit(*visitor*)

setup
Keyword used as the setup or `None` if no setup.

Can be set to a new setup keyword or `None` since RF 3.0.1.

teardown
Keyword used as the teardown or `None` if no teardown.

Can be set to a new teardown keyword or `None` since RF 3.0.1.

all
Iterates over all keywords, including setup and teardown.

normal
Iterates over normal keywords, omitting setup and teardown.

robot.model.message module

class robot.model.message.**Message**(*message=''*, *level='INFO'*, *html=False*, *timestamp=None*, *parent=None*)
Bases: *robot.model.modelobject.ModelObject*

A message created during the test execution.

Can be a log message triggered by a keyword, or a warning or an error that occurred during parsing or test execution.

message
The message content as a string.

level
Severity of the message. Either `TRACE`, `DEBUG`, `INFO`, `WARN`, `ERROR`, or `FAIL`. The latest one is only used with keyword failure messages.

html
`True` if the content is in HTML, `False` otherwise.

timestamp
Timestamp in format `%Y%m%d %H:%M:%S.%f`.

parent
The object this message was triggered by.

html_message
Returns the message content as HTML.

visit(*visitor*)
Visitor interface entry-point.

copy(***attributes*)
Return shallow copy of this object.

Parameters **attributes** – Attributes to be set for the returned copy automatically. For example, `test.copy(name='New name')`.

See also *deepcopy()*. The difference between these two is the same as with the standard `copy.copy` and `copy.deepcopy` functions that these methods also use internally.

New in Robot Framework 3.0.1.

deepcopy(***attributes*)
Return deep copy of this object.

Parameters **attributes** – Attributes to be set for the returned copy automatically. For example, `test.deepcopy(name='New name')`.

See also *copy()*. The difference between these two is the same as with the standard `copy.copy` and `copy.deepcopy` functions that these methods also use internally.

New in Robot Framework 3.0.1.

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*, **start_and_end*)

insert(*index*, *item*)

pop(**index*)

visit(*visitor*)

robot.model.metadata module

class robot.model.metadata.**Metadata**(*initial=None*)
Bases: *robot.utils.normalizing.NormalizedDict*

clear() → None. Remove all items from D.

copy()

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

items() → list of D's (key, value) pairs, as 2-tuples

iteritems() → an iterator over the (key, value) items of D

iterkeys() → an iterator over the keys of D

itervalues() → an iterator over the values of D

keys() → list of D's keys

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

update(*[E]*, ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

values() → list of D's values

robot.model.modelobject module

class robot.model.modelobject.**ModelObject**
Bases: object

copy(***attributes*)
Return shallow copy of this object.

Parameters **attributes** – Attributes to be set for the returned copy automatically. For example, `test.copy(name='New name')`.

See also *deepcopy()*. The difference between these two is the same as with the standard `copy.copy` and `copy.deepcopy` functions that these methods also use internally.

New in Robot Framework 3.0.1.

deepcopy(***attributes*)
Return deep copy of this object.

Parameters **attributes** – Attributes to be set for the returned copy automatically. For example, `test.deepcopy(name='New name')`.

See also *copy()*. The difference between these two is the same as with the standard `copy.copy` and `copy.deepcopy` functions that these methods also use internally.

New in Robot Framework 3.0.1.

robot.model.modifier module

class robot.model.modifier.**ModelModifier**(*visitors*, *empty_suite_ok*, *logger*)
Bases: *robot.model.visitor.SuiteVisitor*

visit_suite(*suite*)
Implements traversing through the suite and its direct children.

Can be overridden to allow modifying the passed in `suite` without calling *start_suite()* or *end_suite()* nor visiting child suites, tests or keywords (setup and teardown) at all.

end_keyword(*keyword*)
Called when keyword ends. Default implementation does nothing.

end_message(*msg*)
Called when message ends. Default implementation does nothing.

end_suite(*suite*)
Called when suite ends. Default implementation does nothing.

end_test(*test*)
Called when test ends. Default implementation does nothing.

start_keyword(*keyword*)
Called when keyword starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_message(*msg*)
Called when message starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_suite(*suite*)
Called when suite starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_test(*test*)
Called when test starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

visit_keyword(*kw*)
Implements traversing through the keyword and its child keywords.

Can be overridden to allow modifying the passed in `kw` without calling *start_keyword()* or *end_keyword()* nor visiting child keywords.

visit_message(*msg*)
Implements visiting the message.

Can be overridden to allow modifying the passed in `msg` without calling *start_message()* or *end_message()*.

visit_test(*test*)
Implements traversing through the test and its keywords.

Can be overridden to allow modifying the passed in `test` without calling *start_test()* or *end_test()* nor visiting keywords.

robot.model.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*)

robot.model.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*, *rpa=False*)
Bases: `object`

Container for total, suite and tag statistics.

Accepted parameters have the same semantics as the matching command line options.

total = None
Instance of *TotalStatistics*.

suite = None
Instance of *SuiteStatistics*.

tags = None
Instance of *TagStatistics*.

visit(*visitor*)

class robot.model.statistics.**StatisticsBuilder**(*total_builder*, *suite_builder*, *tag_builder*)
Bases: *robot.model.visitor.SuiteVisitor*

start_suite(*suite*)
Called when suite starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

end_suite(*suite*)
Called when suite ends. Default implementation does nothing.

visit_test(*test*)
Implements traversing through the test and its keywords.

Can be overridden to allow modifying the passed in `test` without calling *start_test()* or *end_test()* nor visiting keywords.

visit_keyword(*kw*)
Implements traversing through the keyword and its child keywords.

Can be overridden to allow modifying the passed in `kw` without calling *start_keyword()* or *end_keyword()* nor visiting child keywords.

end_keyword(*keyword*)
Called when keyword ends. Default implementation does nothing.

end_message(*msg*)
Called when message ends. Default implementation does nothing.

end_test(*test*)
Called when test ends. Default implementation does nothing.

start_keyword(*keyword*)
Called when keyword starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_message(*msg*)
Called when message starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_test(*test*)
Called when test starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

visit_message(*msg*)
Implements visiting the message.

Can be overridden to allow modifying the passed in `msg` without calling *start_message()* or *end_message()*.

visit_suite(*suite*)
Implements traversing through the suite and its direct children.

Can be overridden to allow modifying the passed in `suite` without calling *start_suite()* or *end_suite()* nor visiting child suites, tests or keywords (setup and teardown) at all.

robot.model.stats module

class robot.model.stats.**Stat**(*name*)
Bases: *robot.utils.sortable.Sortable*

Generic statistic object used for storing all the statistic values.

name = None
Human readable identifier of the object these statistics belong to. Either *All Tests* or *Critical Tests* for *TotalStatistics*, long name of the suite for *SuiteStatistics* or name of the tag for *TagStatistics*

passed = None
Number of passed tests.

failed = None
Number of failed tests.

elapsed = None
Number of milliseconds it took to execute.

get_attributes(*include_label=False*, *include_elapsed=False*, *exclude_empty=True*, *values_as_strings=False*, *html_escape=False*)

total

add_test(*test*)

visit(*visitor*)

class robot.model.stats.**TotalStat**(*name*)
Bases: *robot.model.stats.Stat*

Stores statistic values for a test run.

type = 'total'

add_test(*test*)

get_attributes(*include_label=False*, *include_elapsed=False*, *exclude_empty=True*, *values_as_strings=False*, *html_escape=False*)

total

visit(*visitor*)

class robot.model.stats.**SuiteStat**(*suite*)
Bases: *robot.model.stats.Stat*

Stores statistics values for a single suite.

type = 'suite'

id = None
Identifier of the suite, e.g. *s1-s2*.

elapsed = None
Number of milliseconds it took to execute this suite, including sub-suites.

add_stat(*other*)

add_test(*test*)

get_attributes(*include_label=False*, *include_elapsed=False*, *exclude_empty=True*, *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=None*)

Bases: *robot.model.stats.Stat*

Stores statistic values for a single tag.

type = 'tag'

doc = None

Documentation of tag as a string.

links = None

List of tuples in which the first value is the link URL and the second is the link title. An empty list by default.

critical = None

`True` if tag is considered critical, `False` otherwise.

non_critical = None

`True` if tag is considered non-critical, `False` otherwise.

combined = None

Pattern as a string if the tag is combined, `None` otherwise.

info

Returns additional information of the tag statistics are about. Either *critical*, *non-critical*, *combined* or an empty string.

add_test(*test*)

get_attributes(*include_label=False*, *include_elapsed=False*, *exclude_empty=True*, *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=True*, *values_as_strings=False*, *html_escape=False*)

info

Returns additional information of the tag statistics are about. Either *critical*, *non-critical*, *combined* or an empty string.

total

type = 'tag'

visit(*visitor*)

class robot.model.stats.**CriticalTagStat**(*tag_pattern*, *name=None*, *critical=True*, *doc=''*, *links=None*)

Bases: *robot.model.stats.TagStat*

match(*tags*)

add_test(*test*)

get_attributes(*include_label=False*, *include_elapsed=False*, *exclude_empty=True*, *values_as_strings=False*, *html_escape=False*)

info
Returns additional information of the tag statistics are about. Either *critical*, *non-critical*, *combined* or an empty string.

total

type = 'tag'

visit(*visitor*)

robot.model.suitestatistics module

class robot.model.suitestatistics.**SuiteStatistics**(*suite*)
Bases: object

Container for suite statistics.

stat = None
Instance of *SuiteStat*.

suites = None
List of *TestSuite* objects.

visit(*visitor*)

class robot.model.suitestatistics.**SuiteStatisticsBuilder**(*suite_stat_level*)
Bases: object

current

start_suite(*suite*)

add_test(*test*)

end_suite()

robot.model.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*)

class robot.model.tags.**SingleTagPattern**(*pattern*)
Bases: object

match(*tags*)

class robot.model.tags.**AndTagPattern**(*patterns*)
Bases: object

match(*tags*)

class robot.model.tags.**OrTagPattern**(*patterns*)
Bases: object

match(*tags*)

class robot.model.tags.**NotTagPattern**(*must_match*, **must_not_match*)
Bases: object

match(*tags*)

robot.model.tagsetter module

class robot.model.tagsetter.**TagSetter**(*add=None*, *remove=None*)
Bases: *robot.model.visitor.SuiteVisitor*

start_suite(*suite*)
Called when suite starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

visit_test(*test*)
Implements traversing through the test and its keywords.

Can be overridden to allow modifying the passed in `test` without calling *start_test()* or *end_test()* nor visiting keywords.

visit_keyword(*keyword*)
Implements traversing through the keyword and its child keywords.

Can be overridden to allow modifying the passed in `kw` without calling *start_keyword()* or *end_keyword()* nor visiting child keywords.

end_keyword(*keyword*)
Called when keyword ends. Default implementation does nothing.

end_message(*msg*)
Called when message ends. Default implementation does nothing.

end_suite(*suite*)
Called when suite ends. Default implementation does nothing.

end_test(*test*)
Called when test ends. Default implementation does nothing.

start_keyword(*keyword*)
Called when keyword starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_message(*msg*)
Called when message starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_test(*test*)
Called when test starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

visit_message(*msg*)
Implements visiting the message.

Can be overridden to allow modifying the passed in `msg` without calling *start_message()* or *end_message()*.

visit_suite(*suite*)
Implements traversing through the suite and its direct children.

Can be overridden to allow modifying the passed in `suite` without calling *start_suite()* or *end_suite()* nor visiting child suites, tests or keywords (setup and teardown) at all.

robot.model.tagstatistics module

class robot.model.tagstatistics.**TagStatistics**(*critical_stats*, *non_critical_stats*, *combined_stats*)
Bases: `object`

Container for tag statistics.

tags = None
Dictionary, where key is the name of the tag as a string and value is an instance of *TagStat*.

critical = None
List of *CriticalTagStat* objects.

non_critical = None
List of *CriticalTagStat* objects.

combined = None
List of *CombinedTagStat* objects.

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**(*docs=None*, *links=None*)
Bases: `object`

get_stat(*tag*)

get_critical_stats(*criticality*, *critical=True*)

get_combined_stats(*combined=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*)

robot.model.testcase module

class robot.model.testcase.**TestCase**(*name=''*, *doc=''*, *tags=None*, *timeout=None*)
Bases: *robot.model.modelobject.ModelObject*

Base model for a single test case.

Extended by *robot.running.model.TestCase* and *robot.result.model.TestCase*.

keyword_class
alias of *robot.model.keyword.Keyword*

parent
Parent suite.

name
Test case name.

doc
Test case documentation.

timeout
Test case timeout.

tags
Test tags as a *Tags* object.

keywords
Keywords as a *Keywords* object.

Contains also possible setup and teardown keywords.

id
Test case id in format like `s1-t3`.

See *TestSuite.id* for more information.

longname
Test name prefixed with the long name of the parent suite.

visit(*visitor*)
Visitor interface entry-point.

copy(***attributes*)
Return shallow copy of this object.

Parameters **attributes** – Attributes to be set for the returned copy automatically. For example, `test.copy(name='New name')`.

See also *deepcopy()*. The difference between these two is the same as with the standard `copy.copy` and `copy.deepcopy` functions that these methods also use internally.

New in Robot Framework 3.0.1.

deepcopy(***attributes*)
Return deep copy of this object.

Parameters **attributes** – Attributes to be set for the returned copy automatically. For example, `test.deepcopy(name='New name')`.

See also *copy()*. The difference between these two is the same as with the standard `copy.copy` and `copy.deepcopy` functions that these methods also use internally.

New in Robot Framework 3.0.1.

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*, **start_and_end*)

insert(*index*, *item*)

pop(**index*)

visit(*visitor*)

robot.model.testsuite module

class robot.model.testsuite.**TestSuite**(*name=''*, *doc=''*, *metadata=None*, *source=None*, *rpa=False*)

Bases: *robot.model.modelobject.ModelObject*

Base model for single suite.

Extended by *robot.running.model.TestSuite* and *robot.result.model.TestSuite*.

test_class
alias of *robot.model.testcase.TestCase*

keyword_class
alias of *robot.model.keyword.Keyword*

parent
Parent suite. `None` with the root suite.

doc
Test suite documentation.

source
Path to the source file or directory.

rpa

name
Test suite name. If not set, constructed from child suite names.

longname
Suite name prefixed with the long name of the parent suite.

metadata
Free test suite metadata as a dictionary.

suites
Child suites as a *TestSuites* object.

tests
Tests as a *TestCases* object.

keywords
Suite setup and teardown as a *Keywords* object.

id
An automatically generated unique id.

The root suite has id `s1`, its child suites have ids `s1-s1`, `s1-s2`, ..., their child suites get ids `s1-s1-s1`, `s1-s1-s2`, ..., `s1-s2-s1`, ..., and so on.

The first test in a suite has an id like `s1-t1`, the second has an id `s1-t2`, and so on. Similarly keywords in suites (setup/teardown) and in tests get ids like `s1-k1`, `s1-t1-k1`, and `s1-s4-t2-k5`.

test_count
Number of the tests in this suite, recursively.

set_tags(*add=None*, *remove=None*, *persist=False*)
Add and/or remove specified tags to the tests in this suite.

Parameters

- **add** – Tags to add as a list or, if adding only one, as a single string.
- **remove** – Tags to remove as a list or as a single string. Can be given as patterns where `*` and `?` work as wildcards.
- **persist** – Add/remove specified tags also to new tests added to this suite in the future.

filter(*included_suites=None*, *included_tests=None*, *included_tags=None*, *excluded_tags=None*)
Select test cases and remove others from this suite.

Parameters have the same semantics as `--suite`, `--test`, `--include`, and `--exclude` command line options. All of them can be given as a list of strings, or when selecting only one, as a single string.

Child suites that contain no tests after filtering are automatically removed.

Example:

```
suite.filter(included_tests=['Test 1', '* Example'],
             included_tags='priority-1')
```

configure(***options*)
A shortcut to configure a suite using one method call.

Can only be used with the root test suite.

Parameters **options** – Passed to *SuiteConfigurer* that will then set suite attributes, call *filter()*, etc. as needed.

copy(***attributes*)
Return shallow copy of this object.

Parameters **attributes** – Attributes to be set for the returned copy automatically. For example, `test.copy(name='New name')`.

See also *deepcopy()*. The difference between these two is the same as with the standard `copy.copy` and `copy.deepcopy` functions that these methods also use internally.

New in Robot Framework 3.0.1.

deepcopy(***attributes*)
Return deep copy of this object.

Parameters **attributes** – Attributes to be set for the returned copy automatically. For example, `test.deepcopy(name='New name')`.

See also *copy()*. The difference between these two is the same as with the standard `copy.copy` and `copy.deepcopy` functions that these methods also use internally.

New in Robot Framework 3.0.1.

remove_empty_suites()
Removes all child suites not containing any tests, recursively.

visit(*visitor*)
Visitor interface entry-point.

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*, **start_and_end*)

insert(*index*, *item*)

pop(**index*)

visit(*visitor*)

robot.model.totalstatistics module

class robot.model.totalstatistics.**TotalStatistics**(*rpa=False*)
Bases: `object`

Container for total statistics.

all = None
Instance of *TotalStat* for all the tests.

visit(*visitor*)

message
String representation of the statistics.

For example:

```
2 critical tests, 1 passed, 1 failed
2 tests total, 1 passed, 1 failed
```

class robot.model.totalstatistics.**TotalStatisticsBuilder**(*suite=None*, *rpa=False*)
Bases: *robot.model.visitor.SuiteVisitor*

add_test(*test*)

visit_test(*test*)
Implements traversing through the test and its keywords.

Can be overridden to allow modifying the passed in `test` without calling *start_test()* or *end_test()* nor visiting keywords.

visit_keyword(*kw*)
Implements traversing through the keyword and its child keywords.

Can be overridden to allow modifying the passed in `kw` without calling *start_keyword()* or *end_keyword()* nor visiting child keywords.

end_keyword(*keyword*)
Called when keyword ends. Default implementation does nothing.

end_message(*msg*)
Called when message ends. Default implementation does nothing.

end_suite(*suite*)
Called when suite ends. Default implementation does nothing.

end_test(*test*)
Called when test ends. Default implementation does nothing.

start_keyword(*keyword*)
Called when keyword starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_message(*msg*)
Called when message starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_suite(*suite*)
Called when suite starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_test(*test*)
Called when test starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

visit_message(*msg*)
Implements visiting the message.

Can be overridden to allow modifying the passed in `msg` without calling *start_message()* or *end_message()*.

visit_suite(*suite*)
Implements traversing through the suite and its direct children.

Can be overridden to allow modifying the passed in `suite` without calling *start_suite()* or *end_suite()* nor visiting child suites, tests or keywords (setup and teardown) at all.

robot.model.visitor module

Interface to ease traversing through a test suite structure.

Visitors make it easy to modify test suite structures or to collect information from them. They work both with the *executable model* and the *result model*, but the objects passed to the visitor methods are slightly different depending on the model they are used with. The main differences are that on the execution side keywords do not have child keywords nor messages, and that only the result objects have status related attributes like `status` and `starttime`.

This module contains *SuiteVisitor* that implements the core logic to visit a test suite structure, and the *result* package contains *ResultVisitor* that supports visiting the whole test execution result structure. Both of these visitors should be imported via the *robot.api* package when used by external code.

Visitor algorithm

All suite, test, keyword and message objects have a `visit()` method that accepts a visitor instance. These methods will then call the correct visitor method *visit_suite()*, *visit_test()*, *visit_keyword()* or *visit_message()*, depending on the instance where the `visit()` method exists.

The recommended and definitely easiest way to implement a visitor is extending the *SuiteVisitor* base class. The default implementation of its `visit_x()` methods take care of traversing child elements of the object `x` recursively. A `visit_x()` method first calls a corresponding `start_x()` method (e.g. `visit_suite()` calls `start_suite()`), then calls `visit()` for all child objects of the `x` object, and finally calls the corresponding `end_x()` method. The default implementations of `start_x()` and `end_x()` do nothing.

Visitors extending the *SuiteVisitor* can stop visiting at a certain level either by overriding suitable `visit_x()` method or by returning an explicit `False` from any `start_x()` method.

Examples

The following example visitor modifies the test suite structure it visits. It could be used, for example, with Robot Framework's `--prerunmodifier` option to modify test data before execution.

For more examples it is possible to look at the source code of visitors used internally by Robot Framework itself. Some good examples are *TagSetter* and *keyword removers*.

class robot.model.visitor.**SuiteVisitor**
Bases: `object`

Abstract class to ease traversing through the test suite structure.

See the *module level* documentation for more information and an example.

visit_suite(*suite*)
Implements traversing through the suite and its direct children.

Can be overridden to allow modifying the passed in `suite` without calling *start_suite()* or *end_suite()* nor visiting child suites, tests or keywords (setup and teardown) at all.

start_suite(*suite*)
Called when suite starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

end_suite(*suite*)
Called when suite ends. Default implementation does nothing.

visit_test(*test*)
Implements traversing through the test and its keywords.

Can be overridden to allow modifying the passed in `test` without calling *start_test()* or *end_test()* nor visiting keywords.

start_test(*test*)
Called when test starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

end_test(*test*)
Called when test ends. Default implementation does nothing.

visit_keyword(*kw*)
Implements traversing through the keyword and its child keywords.

Can be overridden to allow modifying the passed in `kw` without calling *start_keyword()* or *end_keyword()* nor visiting child keywords.

start_keyword(*keyword*)
Called when keyword starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

end_keyword(*keyword*)
Called when keyword ends. Default implementation does nothing.

visit_message(*msg*)
Implements visiting the message.

Can be overridden to allow modifying the passed in `msg` without calling *start_message()* or *end_message()*.

start_message(*msg*)
Called when message starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

end_message(*msg*)
Called when message ends. Default implementation does nothing.

robot.output package

Package for internal logging and other output.

Not part of the public API, and also subject to change in the future when test execution is refactored.

Subpackages

robot.output.console package

robot.output.console.**ConsoleOutput**(*type='verbose'*, *width=78*, *colors='AUTO'*, *markers='AUTO'*, *stdout=None*, *stderr=None*)

Submodules

robot.output.console.dotted module

class robot.output.console.dotted.**DottedOutput**(*width=78*, *colors='AUTO'*, *stdout=None*, *stderr=None*)

Bases: `object`

start_suite(*suite*)

end_test(*test*)

end_suite(*suite*)

message(*msg*)

output_file(*name*, *path*)

class robot.output.console.dotted.**StatusReporter**(*stream*, *width*)
Bases: *robot.model.visitor.SuiteVisitor*

report(*suite*)

visit_test(*test*)
Implements traversing through the test and its keywords.

Can be overridden to allow modifying the passed in `test` without calling *start_test()* or *end_test()* nor visiting keywords.

end_keyword(*keyword*)
Called when keyword ends. Default implementation does nothing.

end_message(*msg*)
Called when message ends. Default implementation does nothing.

end_suite(*suite*)
Called when suite ends. Default implementation does nothing.

end_test(*test*)
Called when test ends. Default implementation does nothing.

start_keyword(*keyword*)
Called when keyword starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_message(*msg*)
Called when message starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_suite(*suite*)
Called when suite starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_test(*test*)
Called when test starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

visit_keyword(*kw*)
Implements traversing through the keyword and its child keywords.

Can be overridden to allow modifying the passed in `kw` without calling *start_keyword()* or *end_keyword()* nor visiting child keywords.

visit_message(*msg*)
Implements visiting the message.

Can be overridden to allow modifying the passed in `msg` without calling *start_message()* or *end_message()*.

visit_suite(*suite*)
Implements traversing through the suite and its direct children.

Can be overridden to allow modifying the passed in `suite` without calling *start_suite()* or *end_suite()* nor visiting child suites, tests or keywords (setup and teardown) at all.

robot.output.console.highlighting module

class robot.output.console.highlighting.**HighlightingStream**(*stream*, *colors='AUTO'*)
Bases: object

write(*text*, *flush=True*)

flush()

highlight(*text*, *status=None*, *flush=True*)

error(*message*, *level*)

robot.output.console.highlighting.**Highlighter**(*stream*)

class robot.output.console.highlighting.**AnsiHighlighter**(*stream*)
Bases: object

green()

red()

yellow()

reset()

class robot.output.console.highlighting.**NoHighlighting**(*stream*)
Bases: *robot.output.console.highlighting.AnsiHighlighter*

green()

red()

reset()

yellow()

class robot.output.console.highlighting.**DosHighlighter**(*stream*)
Bases: object

green()

red()

yellow()

reset()

robot.output.console.quiet module

class robot.output.console.quiet.**QuietOutput**(*colors='AUTO'*, *stderr=None*)
Bases: object

message(*msg*)

class robot.output.console.quiet.**NoOutput**
Bases: object

robot.output.console.verbose module

class robot.output.console.verbose.**VerboseOutput**(*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.console.verbose.**VerboseWriter**(*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(*status*)

error(*message*, *level*, *clear=False*)

output(*name*, *path*)

class robot.output.console.verbose.**KeywordMarker**(*highlighter*, *markers*)

Bases: object

mark(*status*)

reset_count()

Submodules

robot.output.debugfile module

robot.output.debugfile.**DebugFile**(*path*)

robot.output.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*)

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*)

robot.output.librarylogger module

Implementation of the public test library logging API.

This is exposed via *robot.api.logger*. Implementation must reside here to avoid cyclic imports.

robot.output.librarylogger.**write**(*msg*, *level*, *html=False*)

robot.output.librarylogger.**trace**(*msg*, *html=False*)

robot.output.librarylogger.**debug**(*msg*, *html=False*)

robot.output.librarylogger.**info**(*msg*, *html=False*, *also_console=False*)

robot.output.librarylogger.**warn**(*msg*, *html=False*)

robot.output.librarylogger.**error**(*msg*, *html=False*)

robot.output.librarylogger.**console**(*msg*, *newline=True*, *stream='stdout'*)

robot.output.listenerarguments module

class robot.output.listenerarguments.**ListenerArguments**(*arguments*)

Bases: object

get_arguments(*version*)

classmethod by_method_name(*name*, *arguments*)

class robot.output.listenerarguments.**MessageArguments**(*arguments*)

Bases: *robot.output.listenerarguments.ListenerArguments*

classmethod by_method_name(*name*, *arguments*)

get_arguments(*version*)

class robot.output.listenerarguments.**StartSuiteArguments**(*arguments*)
Bases: robot.output.listenerarguments._ListenerArgumentsFromItem

classmethod by_method_name(*name*, *arguments*)

get_arguments(*version*)

class robot.output.listenerarguments.**EndSuiteArguments**(*arguments*)
Bases: *robot.output.listenerarguments.StartSuiteArguments*

classmethod by_method_name(*name*, *arguments*)

get_arguments(*version*)

class robot.output.listenerarguments.**StartTestArguments**(*arguments*)
Bases: robot.output.listenerarguments._ListenerArgumentsFromItem

classmethod by_method_name(*name*, *arguments*)

get_arguments(*version*)

class robot.output.listenerarguments.**EndTestArguments**(*arguments*)
Bases: *robot.output.listenerarguments.StartTestArguments*

classmethod by_method_name(*name*, *arguments*)

get_arguments(*version*)

class robot.output.listenerarguments.**StartKeywordArguments**(*arguments*)
Bases: robot.output.listenerarguments._ListenerArgumentsFromItem

classmethod by_method_name(*name*, *arguments*)

get_arguments(*version*)

class robot.output.listenerarguments.**EndKeywordArguments**(*arguments*)
Bases: *robot.output.listenerarguments.StartKeywordArguments*

classmethod by_method_name(*name*, *arguments*)

get_arguments(*version*)

robot.output.listenermethods module

class robot.output.listenermethods.**ListenerMethods**(*method_name*, *listeners*)
Bases: object

class robot.output.listenermethods.**LibraryListenerMethods**(*method_name*)
Bases: object

new_suite_scope()

discard_suite_scope()

register(*listeners*, *library*)

unregister(*library*)

class robot.output.listenermethods.**ListenerMethod**(*method*, *listener*, *library=None*)
Bases: object

called = False

robot.output.listeners module

class robot.output.listeners.**Listeners**(*listeners*, *log_level='INFO'*)
Bases: object

set_log_level(*level*)

log_message(*msg*)

imported(*import_type*, *name*, *attrs*)

output_file(*file_type*, *path*)

class robot.output.listeners.**LibraryListeners**(*log_level='INFO'*)
Bases: object

register(*listeners*, *library*)

unregister(*library*, *close=False*)

new_suite_scope()

discard_suite_scope()

set_log_level(*level*)

log_message(*msg*)

imported(*import_type*, *name*, *attrs*)

output_file(*file_type*, *path*)

class robot.output.listeners.**ListenerProxy**(*listener*, *method_names*, *prefix=None*)
Bases: *robot.output.loggerhelper.AbstractLoggerProxy*

classmethod import_listeners(*listeners*, *method_names*, *prefix=None*, *raise_on_error=False*)

robot.output.logger module

class robot.output.logger.**Logger**(*register_console_logger=True*)
Bases: *robot.output.loggerhelper.AbstractLogger*

A global logger proxy to delegating messages to registered loggers.

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.

NOTE: This API is likely to change in future versions.

start_loggers

end_loggers

register_console_logger(*type='verbose'*, *width=78*, *colors='AUTO'*, *markers='AUTO'*, *stdout=None*, *stderr=None*)

unregister_console_logger()

register_syslog(*path=None*, *level='INFO'*)

register_xml_logger(*logger*)

unregister_xml_logger()

register_listeners(*listeners*, *library_listeners*)

register_logger(**loggers*)

unregister_logger(**loggers*)

disable_message_cache()

register_error_listener(*listener*)

message(*msg*)
Messages about what the framework is doing, warnings, errors, ...

cache_only

delayed_logging

log_message(*msg*)
Messages about what the framework is doing, warnings, errors, ...

log_output(*output*)

enable_library_import_logging()

disable_library_import_logging()

start_suite(*suite*)

end_suite(*suite*)

start_test(*test*)

end_test(*test*)

start_keyword(*keyword*)

end_keyword(*keyword*)

imported(*import_type*, *name*, ***attrs*)

output_file(*file_type*, *path*)
Finished output, report, log, debug, or xunit file

close()

debug(*msg*)

error(*msg*)

fail(*msg*)

info(*msg*)

set_level(*level*)

trace(*msg*)

warn(*msg*)

write(*message*, *level*, *html=False*)

class robot.output.logger.**LoggerProxy**(*logger*, *method_names=None*, *prefix=None*)
Bases: *robot.output.loggerhelper.AbstractLoggerProxy*

robot.output.loggerhelper module

class robot.output.loggerhelper.**AbstractLogger**(*level='TRACE'*)
Bases: object

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*

message

resolve_delayed_message()

copy(***attributes*)

Return shallow copy of this object.

Parameters **attributes** – Attributes to be set for the returned copy automatically. For example, `test.copy(name='New name')`.

See also *deepcopy()*. The difference between these two is the same as with the standard `copy.copy` and `copy.deepcopy` functions that these methods also use internally.

New in Robot Framework 3.0.1.

deepcopy(***attributes*)

Return deep copy of this object.

Parameters **attributes** – Attributes to be set for the returned copy automatically. For example, `test.deepcopy(name='New name')`.

See also *copy()*. The difference between these two is the same as with the standard `copy.copy` and `copy.deepcopy` functions that these methods also use internally.

New in Robot Framework 3.0.1.

html

html_message

Returns the message content as HTML.

level

parent

timestamp

visit(*visitor*)

Visitor interface entry-point.

class robot.output.loggerhelper.**IsLogged**(*level*)

Bases: `object`

set_level(*level*)

class robot.output.loggerhelper.**AbstractLoggerProxy**(*logger*, *method_names=None*, *prefix=None*)

Bases: object

robot.output.output module

class robot.output.output.**Output**(*settings*)

Bases: *robot.output.loggerhelper.AbstractLogger*

register_error_listener(*listener*)

close(*result*)

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*)

robot.output.pyloggingconf module

robot.output.pyloggingconf.**robot_handler_enabled**(**args*, ***kwds*)

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*)

Do whatever it takes to actually log the specified logging record.

This version is intended to be implemented by subclasses and so raises a NotImplementedError.

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*)

robot.output.stdoutlogsplitter module

class robot.output.stdoutlogsplitter.**StdoutLogSplitter**(*output*)
Bases: object

Splits messages logged through stdout (or stderr) into Message objects

robot.output.xmllogger module

class robot.output.xmllogger.**XmlLogger**(*path*, *log_level='TRACE'*, *rpa=False*, *generator='Robot'*)
Bases: *robot.result.visitor.ResultVisitor*

close()

set_log_level(*level*)

message(*msg*)

log_message(*msg*)

start_keyword(*kw*)
Called when keyword starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

end_keyword(*kw*)
Called when keyword ends. Default implementation does nothing.

start_test(*test*)
Called when test starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

end_test(*test*)
Called when test ends. Default implementation does nothing.

start_suite(*suite*)
Called when suite starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

end_suite(*suite*)
Called when suite ends. Default implementation does nothing.

start_statistics(*stats*)

end_statistics(*stats*)

start_total_statistics(*total_stats*)

end_total_statistics(*total_stats*)

start_tag_statistics(*tag_stats*)

end_tag_statistics(*tag_stats*)

start_suite_statistics(*tag_stats*)

end_suite_statistics(*tag_stats*)

visit_stat(*stat*)

start_errors(*errors=None*)

end_errors(*errors=None*)

end_message(*msg*)
Called when message ends. Default implementation does nothing.

end_result(*result*)

end_stat(*stat*)

start_message(*msg*)
Called when message starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_result(*result*)

start_stat(*stat*)

visit_errors(*errors*)

visit_keyword(*kw*)
Implements traversing through the keyword and its child keywords.

Can be overridden to allow modifying the passed in `kw` without calling *start_keyword()* or *end_keyword()* nor visiting child keywords.

visit_message(*msg*)
Implements visiting the message.

Can be overridden to allow modifying the passed in `msg` without calling *start_message()* or *end_message()*.

visit_result(*result*)

visit_statistics(*stats*)

visit_suite(*suite*)
Implements traversing through the suite and its direct children.

Can be overridden to allow modifying the passed in `suite` without calling *start_suite()* or *end_suite()* nor visiting child suites, tests or keywords (setup and teardown) at all.

visit_suite_statistics(*stats*)

visit_tag_statistics(*stats*)

visit_test(*test*)
Implements traversing through the test and its keywords.

Can be overridden to allow modifying the passed in `test` without calling *start_test()* or *end_test()* nor visiting keywords.

visit_total_statistics(*stats*)

robot.parsing package

Implements test data parsing.

Classes *TestCaseFile*, *TestDataDirectory* and *ResourceFile* represented parsed test data. Objects of these classes can be modified and saved back to disk. In addition, a convenience factory function *TestData()* can be used to parse a test case file or directory to a corresponding object.

Aforementioned classes and functions are part of the public API. It is recommended that they are imported through the *robot.api* package like in the example below.

This package is likely to change radically in Robot Framework 2.9. The main motivation for the planned changes is making the data easier to use for external tools that use these modules.

Example

```
import sys
from robot.api import TestData

def print_suite(suite):
    print 'Suite:', suite.name
    for test in suite.testcase_table:
        print '-', test.name
    for child in suite.children:
        print_suite(child)

suite = TestData(source=sys.argv[1])
print_suite(suite)
```

robot.parsing.**disable_curdir_processing**(*method*)
Decorator to disable processing *${CURDIR}* variable.

Submodules

robot.parsing.comments module

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()

robot.parsing.datarow module

class robot.parsing.datarow.**DataRow**(*cells*, *source=None*)
Bases: object

head

tail

all

data

dedent()

starts_for_loop()

starts_test_or_user_keyword_setting()

test_or_user_keyword_setting_name()

is_indented()

is_continuing()

is_commented()

robot.parsing.htmlreader module

class robot.parsing.htmlreader.**HtmlReader**

Bases: HTMLParser.HTMLParser

IGNORE = 0

INITIAL = 1

PROCESS = 2

read(*htmlfile*, *populator*, *path=None*)

handle_starttag(*tag*, *attrs*)

handle_endtag(*tag*)

handle_data(*data*)

handle_entityref(*name*)

handle_charref(*number*)

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*)

handle_pi(*data*)

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*)

robot.parsing.model module

robot.parsing.model.**TestData**(*parent=None*, *source=None*, *include_suites=None*, *warn_on_skipped='DEPRECATED'*, *extensions=None*)
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.
- **warn_on_skipped** – Deprecated.
- **extensions** – List/set of extensions to parse. If None, all files supported by Robot Framework are parsed when searching test cases.

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.

Parameters

- **parent** – parent object to be used in creation of the model object.
- **source** – path where test data is read from.

populate()

has_tests()

imports

keywords

name

report_invalid_syntax(*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.

Parameters **source** – path where resource file is read from.

populate()

imports

keywords

name

report_invalid_syntax(*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 hiearchical structure of other *TestDataDirectory* and *TestCaseFile* objects.

Parameters

- **parent** – parent object to be used in creation of the model object.

- **source** – path where test data is read from.

populate(*include_suites=None*, *extensions=None*, *recurse=True*)

add_child(*path*, *include_suites*, *extensions=None*)

has_tests()

imports

keywords

name

report_invalid_syntax(*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.**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(*name*)

header

name

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(*name*)

header

name

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(*name*)

header

name

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'

set_header(*header*)

add(*name*)

is_started()

directory

header

name

report_invalid_syntax(*message*, *level='ERROR'*)

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**(*parent*, *name*, *value*, *comment=None*)
Bases: object

as_list()

is_set()

is_for_loop()

has_data()

report_invalid_syntax(*message*, *level='ERROR'*)

class robot.parsing.model.**TestCase**(*parent*, *name*)
Bases: robot.parsing.model._WithSteps, robot.parsing.model._WithSettings

source

directory

add_for_loop(*declaration*, *comment=None*)

end_for_loop()

report_invalid_syntax(*message*, *level='ERROR'*)

settings

add_step(*content*, *comment=None*)

copy(*name*)

get_setter(*name*)

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

end_for_loop()

get_setter(*name*)

report_invalid_syntax(*message*, *level='ERROR'*)

source

class robot.parsing.model.**ForLoop**(*parent*, *declaration*, *comment=None*)
Bases: robot.parsing.model._WithSteps

The parsed representation of a for-loop.

Parameters

- **declaration** (*list*) – The literal cell values that declare the loop (excluding ":FOR").
- **comment** (*str*) – A comment, default None.

Variables

- ***flavor*** (*str*) – The value of the 'IN' item, uppercased. Typically 'IN', 'IN RANGE', 'IN ZIP', or 'IN ENUMERATE'.
- **vars** (*list*) – Variables set per-iteration by this loop.
- ***items*** (*list*) – Loop values that come after the 'IN' item.
- ***comment*** (*str*) – A comment, or None.
- **steps** (*list*) – A list of steps in the loop.

flavors = set(['IN', 'IN ENUMERATE', 'IN RANGE', 'IN ZIP'])

normalized_flavors = <robot.utils.normalizing.NormalizedDict object>

is_comment()

is_for_loop()

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()

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*)

robot.parsing.populators module

exception robot.parsing.populators.**NoTestsFound**(*message=''*, *details=''*)
Bases: *robot.errors.DataError*

args

message

class robot.parsing.populators.**FromFilePopulator**(*datafile*)
Bases: object

populate(*path*, *resource=False*)

start_table(*header*)

eof()

add(*row*)

class robot.parsing.populators.**FromDirectoryPopulator**
Bases: object

ignored_prefixes = ('_', '.')

ignored_dirs = ('CVS',)

populate(*path*, *datadir*, *include_suites=None*, *include_extensions=None*, *recurse=True*)

robot.parsing.restreader module

robot.parsing.restreader.**RestReader**()

robot.parsing.restsupport module

class robot.parsing.restsupport.**CaptureRobotData**(*name*, *arguments*, *options*, *content*, *lineno*, *content_offset*, *block_text*, *state*, *state_machine*)
Bases: docutils.parsers.rst.directives.body.CodeBlock

run()

add_name(*node*)
Append self.options['name'] to node['names'] if it exists.

Also normalize the name string and register it as explicit target.

assert_has_content()
Throw an ERROR-level DirectiveError if the directive doesn't have contents.

debug(*message*)

directive_error(*level*, *message*)
Return a DirectiveError suitable for being thrown as an exception.

Call "raise self.directive_error(level, message)" from within a directive implementation to return one single system message at level *level*, which automatically gets the directive block and the line number added.

Preferably use the *debug*, *info*, *warning*, *error*, or *severe* wrapper methods, e.g. self.error(message) to generate an ERROR-level directive error.

error(*message*)

final_argument_whitespace = False

has_content = True

info(*message*)

option_spec = {'class': <function class_option>, 'name': <function unchanged>, 'numb

optional_arguments = 1

required_arguments = 0

severe(*message*)

warning(*message*)

class robot.parsing.restsupport.**RobotDataStorage**(*doctree*)
Bases: object

add_data(*rows*)

get_data()

has_data()

robot.parsing.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()

is_active()

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*

keyword

is_comment()

is_set()

is_active()

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'*, *parent=None*)

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'*, *parent=None*)

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'*, *parent=None*)

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*)

robot.parsing.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*)

robot.parsing.tsvreader module

class robot.parsing.tsvreader.**TsvReader**
Bases: robot.parsing.robotreader.RobotReader

classmethod split_row(*row*)

read(*file*, *populator*, *path=None*)

robot.parsing.txtreader module

robot.reporting package

Implements report, log, output XML, and xUnit file generation.

The public API of this package is the *ResultWriter* class. It can write result files based on XML output files on the file system, as well as based on the result objects returned by the *ExecutionResult()* factory method or an executed *TestSuite*.

It is highly recommended to use the public API via the *robot.api* package.

This package is considered stable.

Submodules

robot.reporting.jsbuildingcontext module

class robot.reporting.jsbuildingcontext.**JsBuildingContext**(*log_path=None*, *split_log=False*, *prune_input=False*)
Bases: object

string(*string*, *escape=True*, *attr=False*)

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(***kwds*)

robot.reporting.jsexecutionresult module

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()

robot.reporting.jsmodelbuilders module

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*)

robot.reporting.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*)

robot.reporting.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*)

robot.reporting.outputwriter module

class robot.reporting.outputwriter.**OutputWriter**(*output*, *rpa=False*)

Bases: *robot.output.xmllogger.XmlLogger*

start_message(*msg*)

Called when message starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

visit_keyword(*kw*)

Implements traversing through the keyword and its child keywords.

Can be overridden to allow modifying the passed in `kw` without calling *start_keyword()* or *end_keyword()* nor visiting child keywords.

close()

end_result(*result*)

end_errors(*errors=None*)

end_keyword(*kw*)
Called when keyword ends. Default implementation does nothing.

end_message(*msg*)
Called when message ends. Default implementation does nothing.

end_stat(*stat*)

end_statistics(*stats*)

end_suite(*suite*)
Called when suite ends. Default implementation does nothing.

end_suite_statistics(*tag_stats*)

end_tag_statistics(*tag_stats*)

end_test(*test*)
Called when test ends. Default implementation does nothing.

end_total_statistics(*total_stats*)

log_message(*msg*)

message(*msg*)

set_log_level(*level*)

start_errors(*errors=None*)

start_keyword(*kw*)
Called when keyword starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_result(*result*)

start_stat(*stat*)

start_statistics(*stats*)

start_suite(*suite*)
Called when suite starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_suite_statistics(*tag_stats*)

start_tag_statistics(*tag_stats*)

start_test(*test*)
Called when test starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_total_statistics(*total_stats*)

visit_errors(*errors*)

visit_message(*msg*)
Implements visiting the message.

Can be overridden to allow modifying the passed in `msg` without calling *start_message()* or *end_message()*.

visit_result(*result*)

visit_stat(*stat*)

visit_statistics(*stats*)

visit_suite(*suite*)
Implements traversing through the suite and its direct children.

Can be overridden to allow modifying the passed in `suite` without calling *`start_suite()`* or *`end_suite()`* nor visiting child suites, tests or keywords (setup and teardown) at all.

visit_suite_statistics(*stats*)

visit_tag_statistics(*stats*)

visit_test(*test*)
Implements traversing through the test and its keywords.

Can be overridden to allow modifying the passed in `test` without calling *`start_test()`* or *`end_test()`* nor visiting keywords.

visit_total_statistics(*stats*)

robot.reporting.resultwriter module

class robot.reporting.resultwriter.**ResultWriter**(**sources*)
Bases: `object`

A class to create log, report, output XML and xUnit files.

Parameters **sources** – Either one *`Result`* object, or one or more paths to existing output XML files.

By default writes `report.html` and `log.html`, but no output XML or xUnit files. Custom file names can be given and results disabled or enabled using `settings` or `options` passed to the *`write_results()`* method. The latter is typically more convenient:

```
writer = ResultWriter(result)
writer.write_results(report='custom.html', log=None, xunit='xunit.xml')
```

write_results(*settings=None*, ***options*)
Writes results based on the given `settings` or `options`.

Parameters

- **settings** – *`RebotSettings`* object to configure result writing.
- **options** – Used to construct new *`RebotSettings`* object if `settings` are not given.

class robot.reporting.resultwriter.**Results**(*settings*, **sources*)
Bases: `object`

result

js_result

robot.reporting.stringcache module

class robot.reporting.stringcache.**StringIndex**
Bases: `int`

bit_length() → int
Number of bits necessary to represent self in binary. >>> bin(37) '0b100101' >>> (37).bit_length() 6

conjugate()
Returns self, the complex conjugate of any int.

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()

robot.reporting.xunitwriter module

class robot.reporting.xunitwriter.**XUnitWriter**(*execution_result*, *skip_noncritical*)
Bases: object

write(*output*)

class robot.reporting.xunitwriter.**XUnitFileWriter**(*xml_writer*, *skip_noncritical=False*)
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*)
Called when suite starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

end_suite(*suite*)
Called when suite ends. Default implementation does nothing.

visit_test(*test*)
Implements traversing through the test and its keywords.

Can be overridden to allow modifying the passed in `test` without calling *start_test()* or *end_test()* nor visiting keywords.

visit_keyword(*kw*)
Implements traversing through the keyword and its child keywords.

Can be overridden to allow modifying the passed in `kw` without calling *start_keyword()* or *end_keyword()* nor visiting child keywords.

visit_statistics(*stats*)

visit_errors(*errors*)

end_result(*result*)

end_errors(*errors*)

end_keyword(*keyword*)
Called when keyword ends. Default implementation does nothing.

end_message(*msg*)
Called when message ends. Default implementation does nothing.

end_stat(*stat*)

end_statistics(*stats*)

end_suite_statistics(*suite_stats*)

end_tag_statistics(*stats*)

end_test(*test*)
Called when test ends. Default implementation does nothing.

end_total_statistics(*stats*)

start_errors(*errors*)

start_keyword(*keyword*)
Called when keyword starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_message(*msg*)
Called when message starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_result(*result*)

start_stat(*stat*)

start_statistics(*stats*)

start_suite_statistics(*stats*)

start_tag_statistics(*stats*)

start_test(*test*)
Called when test starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_total_statistics(*stats*)

visit_message(*msg*)
Implements visiting the message.

Can be overridden to allow modifying the passed in `msg` without calling *start_message()* or *end_message()*.

visit_result(*result*)

visit_stat(*stat*)

visit_suite(*suite*)
Implements traversing through the suite and its direct children.

Can be overridden to allow modifying the passed in `suite` without calling *start_suite()* or *end_suite()* nor visiting child suites, tests or keywords (setup and teardown) at all.

visit_suite_statistics(*stats*)

visit_tag_statistics(*stats*)

visit_total_statistics(*stats*)

robot.result package

Implements parsing execution results from XML output files.

The main public API of this package consists of the *ExecutionResult()* factory method, that returns *Result* objects, and of the *ResultVisitor* abstract class, that eases further processing the results.

The model objects in the *model* module can also be considered to be part of the public API, because they can be found inside the *Result* object. They can also be inspected and modified as part of the normal test execution by pre-Rebot modifiers and listeners.

It is highly recommended to import the public entry-points via the *robot.api* package like in the example below. In those rare cases where the aforementioned model objects are needed directly, they can be imported from this package.

This package is considered stable.

Example

```
#!/usr/bin/env python

"""Usage: check_test_times.py seconds inpath [outpath]

Reads test execution result from an output XML file and checks that no test
took longer than given amount of seconds to execute.

Optional `outpath` specifies where to write processed results. If not given,
results are written over the original file.
"""

import sys
from robot.api import ExecutionResult, ResultVisitor


class ExecutionTimeChecker(ResultVisitor):

    def __init__(self, max_seconds):
        self.max_milliseconds = max_seconds * 1000

    def visit_test(self, test):
        if test.status == 'PASS' and test.elapsedtime > self.max_milliseconds:
            test.status = 'FAIL'
            test.message = 'Test execution took too long.'


def check_tests(seconds, inpath, outpath=None):
    result = ExecutionResult(inpath)
    result.visit(ExecutionTimeChecker(float(seconds)))
    result.save(outpath)


if __name__ == '__main__':
    try:
        check_tests(*sys.argv[1:])
```

(continues on next page)

(continued from previous page)

```
    except TypeError:
        print __doc__
```

Submodules

robot.result.configurer module

class robot.result.configurer.**SuiteConfigurer**(*remove_keywords=None*, *log_level=None*, *start_time=None*, *end_time=None*, *critical_tags=None*, *non_critical_tags=None*, ***base_config*)

Bases: *robot.model.configurer.SuiteConfigurer*

Result suite configured.

Calls suite's `remove_keywords()`, `filter_messages()` and `set_criticality()` methods and sets its start and end time based on the given named parameters.

`base_config` is forwarded to *robot.model.SuiteConfigurer* that will do further configuration based on them.

visit_suite(*suite*)
Implements traversing through the suite and its direct children.

Can be overridden to allow modifying the passed in `suite` without calling *start_suite()* or *end_suite()* nor visiting child suites, tests or keywords (setup and teardown) at all.

add_tags

end_keyword(*keyword*)
Called when keyword ends. Default implementation does nothing.

end_message(*msg*)
Called when message ends. Default implementation does nothing.

end_suite(*suite*)
Called when suite ends. Default implementation does nothing.

end_test(*test*)
Called when test ends. Default implementation does nothing.

remove_tags

start_keyword(*keyword*)
Called when keyword starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_message(*msg*)
Called when message starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_suite(*suite*)
Called when suite starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_test(*test*)
Called when test starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

visit_keyword(*kw*)
Implements traversing through the keyword and its child keywords.

Can be overridden to allow modifying the passed in `kw` without calling *start_keyword()* or *end_keyword()* nor visiting child keywords.

visit_message(*msg*)
Implements visiting the message.

Can be overridden to allow modifying the passed in `msg` without calling *start_message()* or *end_message()*.

visit_test(*test*)
Implements traversing through the test and its keywords.

Can be overridden to allow modifying the passed in `test` without calling *start_test()* or *end_test()* nor visiting keywords.

robot.result.executionerrors module

class robot.result.executionerrors.**ExecutionErrors**(*messages=None*)
Bases: `object`

Represents errors occurred during the execution of tests.

An error might be, for example, that importing a library has failed.

message_class
alias of *robot.result.model.Message*

messages
A *list-like object* of *Message* instances.

add(*other*)

visit(*visitor*)

robot.result.executionresult module

class robot.result.executionresult.**Result**(*source=None*, *root_suite=None*, *errors=None*, *rpa=None*)
Bases: `object`

Test execution results.

Can be created based on XML output files using the *ExecutionResult()* factory method. Also returned by the *robot.running.TestSuite.run* method.

source = None
Path to the XML file where results are read from.

suite = None
Hierarchical execution results as a *TestSuite* object.

errors = None
Execution errors as an *ExecutionErrors* object.

statistics
Test execution statistics.

Statistics are an instance of *Statistics* that is created based on the contained `suite` and possible *configuration*.

Statistics are created every time this property is accessed. Saving them to a variable is thus often a good idea to avoid re-creating them unnecessarily:

```
from robot.api import ExecutionResult

result = ExecutionResult('output.xml')
result.configure(stat_config={'suite_stat_level': 2,
                              'tag_stat_combine': 'tagANDanother'})
stats = result.statistics
print stats.total.critical.failed
print stats.total.critical.passed
print stats.tags.combined[0].total
```

return_code
Return code (integer) of test execution.

By default returns the number of failed critical tests (max 250), but can be *configured* to always return 0.

configure(*status_rc=True*, *suite_config=None*, *stat_config=None*)
Configures the result object and objects it contains.

Parameters

- **status_rc** – If set to `False`, *return_code* always returns 0.
- **suite_config** – A dictionary of configuration options passed to `configure()` method of the contained `suite`.
- **stat_config** – A dictionary of configuration options used when creating *statistics*.

save(*path=None*)
Save results as a new output XML file.

Parameters **path** – Path to save results to. If omitted, overwrites the original file.

visit(*visitor*)
An entry point to visit the whole result object.

Parameters **visitor** – An instance of *ResultVisitor*.

Visitors can gather information, modify results, etc. See *result* package for a simple usage example.

Notice that it is also possible to call `result.suite.visit` if there is no need to visit the contained `statistics` or `errors`.

handle_suite_teardown_failures()
Internal usage only.

set_execution_mode(*other*)
Set execution mode based on other result. Internal usage only.

class robot.result.executionresult.**CombinedResult**(*results=None*)
Bases: *robot.result.executionresult.Result*

Combined results of multiple test executions.

add_result(*other*)

configure(*status_rc=True*, *suite_config=None*, *stat_config=None*)
Configures the result object and objects it contains.

Parameters

- **status_rc** – If set to `False`, *return_code* always returns 0.
- **suite_config** – A dictionary of configuration options passed to `configure()` method of the contained `suite`.
- **stat_config** – A dictionary of configuration options used when creating *statistics*.

handle_suite_teardown_failures()
Internal usage only.

return_code
Return code (integer) of test execution.

By default returns the number of failed critical tests (max 250), but can be *configured* to always return 0.

save(*path=None*)
Save results as a new output XML file.

Parameters **path** – Path to save results to. If omitted, overwrites the original file.

set_execution_mode(*other*)
Set execution mode based on other result. Internal usage only.

statistics
Test execution statistics.

Statistics are an instance of *Statistics* that is created based on the contained `suite` and possible *configuration*.

Statistics are created every time this property is accessed. Saving them to a variable is thus often a good idea to avoid re-creating them unnecessarily:

```
from robot.api import ExecutionResult

result = ExecutionResult('output.xml')
result.configure(stat_config={'suite_stat_level': 2,
                              'tag_stat_combine': 'tagANDanother'})
stats = result.statistics
print stats.total.critical.failed
print stats.total.critical.passed
print stats.tags.combined[0].total
```

visit(*visitor*)
An entry point to visit the whole result object.

Parameters **visitor** – An instance of *ResultVisitor*.

Visitors can gather information, modify results, etc. See *result* package for a simple usage example.

Notice that it is also possible to call `result.suite.visit` if there is no need to visit the contained `statistics` or `errors`.

robot.result.flattenkeywordmatcher module

robot.result.flattenkeywordmatcher.**validate_flatten_keyword**(*options*)

class robot.result.flattenkeywordmatcher.**FlattenByTypeMatcher**(*flatten*)
Bases: object

match(*kwtype*)

class robot.result.flattenkeywordmatcher.**FlattenByNameMatcher**(*flatten*)
Bases: object

match(*kwname*, *libname=None*)

class robot.result.flattenkeywordmatcher.**FlattenByTagMatcher**(*flatten*)
Bases: object

match(*kwtags*)

robot.result.keywordremover module

robot.result.keywordremover.**KeywordRemover**(*how*)

class robot.result.keywordremover.**AllKeywordsRemover**
Bases: robot.result.keywordremover._KeywordRemover

visit_keyword(*keyword*)
Implements traversing through the keyword and its child keywords.

Can be overridden to allow modifying the passed in `kw` without calling *start_keyword()* or *end_keyword()* nor visiting child keywords.

end_keyword(*keyword*)
Called when keyword ends. Default implementation does nothing.

end_message(*msg*)
Called when message ends. Default implementation does nothing.

end_suite(*suite*)
Called when suite ends. Default implementation does nothing.

end_test(*test*)
Called when test ends. Default implementation does nothing.

start_keyword(*keyword*)
Called when keyword starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_message(*msg*)
Called when message starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_suite(*suite*)
Called when suite starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_test(*test*)
Called when test starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

visit_message(*msg*)
Implements visiting the message.

Can be overridden to allow modifying the passed in `msg` without calling *start_message()* or *end_message()*.

visit_suite(*suite*)
Implements traversing through the suite and its direct children.

Can be overridden to allow modifying the passed in `suite` without calling *start_suite()* or *end_suite()* nor visiting child suites, tests or keywords (setup and teardown) at all.

visit_test(*test*)
Implements traversing through the test and its keywords.

Can be overridden to allow modifying the passed in `test` without calling *start_test()* or *end_test()* nor visiting keywords.

class robot.result.keywordremover.**PassedKeywordRemover**
Bases: robot.result.keywordremover._KeywordRemover

start_suite(*suite*)
Called when suite starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

visit_test(*test*)
Implements traversing through the test and its keywords.

Can be overridden to allow modifying the passed in `test` without calling *start_test()* or *end_test()* nor visiting keywords.

visit_keyword(*keyword*)
Implements traversing through the keyword and its child keywords.

Can be overridden to allow modifying the passed in `kw` without calling *start_keyword()* or *end_keyword()* nor visiting child keywords.

end_keyword(*keyword*)
Called when keyword ends. Default implementation does nothing.

end_message(*msg*)
Called when message ends. Default implementation does nothing.

end_suite(*suite*)
Called when suite ends. Default implementation does nothing.

end_test(*test*)
Called when test ends. Default implementation does nothing.

start_keyword(*keyword*)
Called when keyword starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_message(*msg*)
Called when message starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_test(*test*)
Called when test starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

visit_message(*msg*)
Implements visiting the message.

Can be overridden to allow modifying the passed in `msg` without calling *start_message()* or *end_message()*.

visit_suite(*suite*)
Implements traversing through the suite and its direct children.

Can be overridden to allow modifying the passed in `suite` without calling *start_suite()* or *end_suite()* nor visiting child suites, tests or keywords (setup and teardown) at all.

class robot.result.keywordremover.**ByNameKeywordRemover**(*pattern*)
Bases: robot.result.keywordremover._KeywordRemover

start_keyword(*kw*)
Called when keyword starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

end_keyword(*keyword*)
Called when keyword ends. Default implementation does nothing.

end_message(*msg*)
Called when message ends. Default implementation does nothing.

end_suite(*suite*)
Called when suite ends. Default implementation does nothing.

end_test(*test*)
Called when test ends. Default implementation does nothing.

start_message(*msg*)
Called when message starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_suite(*suite*)
Called when suite starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_test(*test*)
Called when test starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

visit_keyword(*kw*)
Implements traversing through the keyword and its child keywords.

Can be overridden to allow modifying the passed in `kw` without calling *start_keyword()* or *end_keyword()* nor visiting child keywords.

visit_message(*msg*)
Implements visiting the message.

Can be overridden to allow modifying the passed in `msg` without calling *start_message()* or *end_message()*.

visit_suite(*suite*)
Implements traversing through the suite and its direct children.

Can be overridden to allow modifying the passed in `suite` without calling *start_suite()* or *end_suite()* nor visiting child suites, tests or keywords (setup and teardown) at all.

visit_test(*test*)
Implements traversing through the test and its keywords.

Can be overridden to allow modifying the passed in `test` without calling *start_test()* or *end_test()* nor visiting keywords.

class robot.result.keywordremover.**ByTagKeywordRemover**(*pattern*)
Bases: robot.result.keywordremover._KeywordRemover

start_keyword(*kw*)
Called when keyword starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

end_keyword(*keyword*)
Called when keyword ends. Default implementation does nothing.

end_message(*msg*)
Called when message ends. Default implementation does nothing.

end_suite(*suite*)
Called when suite ends. Default implementation does nothing.

end_test(*test*)
Called when test ends. Default implementation does nothing.

start_message(*msg*)
Called when message starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_suite(*suite*)
Called when suite starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_test(*test*)
Called when test starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

visit_keyword(*kw*)
Implements traversing through the keyword and its child keywords.

Can be overridden to allow modifying the passed in `kw` without calling *start_keyword()* or *end_keyword()* nor visiting child keywords.

visit_message(*msg*)
Implements visiting the message.

Can be overridden to allow modifying the passed in `msg` without calling *start_message()* or *end_message()*.

visit_suite(*suite*)
Implements traversing through the suite and its direct children.

Can be overridden to allow modifying the passed in `suite` without calling *start_suite()* or *end_suite()* nor visiting child suites, tests or keywords (setup and teardown) at all.

visit_test(*test*)
Implements traversing through the test and its keywords.

Can be overridden to allow modifying the passed in `test` without calling *start_test()* or *end_test()* nor visiting keywords.

class robot.result.keywordremover.**ForLoopItemsRemover**
Bases: robot.result.keywordremover._KeywordRemover

start_keyword(*kw*)
Called when keyword starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

end_keyword(*keyword*)
Called when keyword ends. Default implementation does nothing.

end_message(*msg*)
Called when message ends. Default implementation does nothing.

end_suite(*suite*)
Called when suite ends. Default implementation does nothing.

end_test(*test*)
Called when test ends. Default implementation does nothing.

start_message(*msg*)
Called when message starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_suite(*suite*)
Called when suite starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_test(*test*)
Called when test starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

visit_keyword(*kw*)
Implements traversing through the keyword and its child keywords.

Can be overridden to allow modifying the passed in `kw` without calling *start_keyword()* or *end_keyword()* nor visiting child keywords.

visit_message(*msg*)
Implements visiting the message.

Can be overridden to allow modifying the passed in `msg` without calling *start_message()* or *end_message()*.

visit_suite(*suite*)
Implements traversing through the suite and its direct children.

Can be overridden to allow modifying the passed in `suite` without calling *start_suite()* or *end_suite()* nor visiting child suites, tests or keywords (setup and teardown) at all.

visit_test(*test*)
Implements traversing through the test and its keywords.

Can be overridden to allow modifying the passed in `test` without calling *start_test()* or *end_test()* nor visiting keywords.

class robot.result.keywordremover.**WaitUntilKeywordSucceedsRemover**
Bases: robot.result.keywordremover._KeywordRemover

start_keyword(*kw*)
Called when keyword starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

end_keyword(*keyword*)
Called when keyword ends. Default implementation does nothing.

end_message(*msg*)
Called when message ends. Default implementation does nothing.

end_suite(*suite*)
Called when suite ends. Default implementation does nothing.

end_test(*test*)
Called when test ends. Default implementation does nothing.

start_message(*msg*)
Called when message starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_suite(*suite*)
Called when suite starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_test(*test*)
Called when test starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

visit_keyword(*kw*)
Implements traversing through the keyword and its child keywords.

Can be overridden to allow modifying the passed in `kw` without calling *start_keyword()* or *end_keyword()* nor visiting child keywords.

visit_message(*msg*)
Implements visiting the message.

Can be overridden to allow modifying the passed in `msg` without calling *start_message()* or *end_message()*.

visit_suite(*suite*)
Implements traversing through the suite and its direct children.

Can be overridden to allow modifying the passed in `suite` without calling *start_suite()* or *end_suite()* nor visiting child suites, tests or keywords (setup and teardown) at all.

visit_test(*test*)
Implements traversing through the test and its keywords.

Can be overridden to allow modifying the passed in `test` without calling *start_test()* or *end_test()* nor visiting keywords.

class robot.result.keywordremover.**WarningAndErrorFinder**
Bases: *robot.model.visitor.SuiteVisitor*

start_suite(*suite*)
Called when suite starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_test(*test*)
Called when test starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_keyword(*keyword*)
Called when keyword starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

visit_message(*msg*)
Implements visiting the message.

Can be overridden to allow modifying the passed in `msg` without calling *start_message()* or *end_message()*.

end_keyword(*keyword*)
Called when keyword ends. Default implementation does nothing.

end_message(*msg*)
Called when message ends. Default implementation does nothing.

end_suite(*suite*)
Called when suite ends. Default implementation does nothing.

end_test(*test*)
Called when test ends. Default implementation does nothing.

start_message(*msg*)
Called when message starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

visit_keyword(*kw*)
Implements traversing through the keyword and its child keywords.

Can be overridden to allow modifying the passed in `kw` without calling *start_keyword()* or *end_keyword()* nor visiting child keywords.

visit_suite(*suite*)
Implements traversing through the suite and its direct children.

Can be overridden to allow modifying the passed in `suite` without calling *start_suite()* or *end_suite()* nor visiting child suites, tests or keywords (setup and teardown) at all.

visit_test(*test*)
Implements traversing through the test and its keywords.

Can be overridden to allow modifying the passed in `test` without calling *start_test()* or *end_test()* nor visiting keywords.

class robot.result.keywordremover.**RemovalMessage**(*message*)
Bases: `object`

set_if_removed(*kw*, *len_before*)

set(*kw*, *message=None*)

robot.result.merger module

class robot.result.merger.**Merger**(*result*)
Bases: *robot.model.visitor.SuiteVisitor*

merge(*merged*)

start_suite(*suite*)
Called when suite starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

end_suite(*suite*)
Called when suite ends. Default implementation does nothing.

visit_test(*test*)
Implements traversing through the test and its keywords.

Can be overridden to allow modifying the passed in `test` without calling *start_test()* or *end_test()* nor visiting keywords.

end_keyword(*keyword*)
Called when keyword ends. Default implementation does nothing.

end_message(*msg*)
Called when message ends. Default implementation does nothing.

end_test(*test*)
Called when test ends. Default implementation does nothing.

start_keyword(*keyword*)
Called when keyword starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_message(*msg*)
Called when message starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_test(*test*)
Called when test starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

visit_keyword(*kw*)
Implements traversing through the keyword and its child keywords.

Can be overridden to allow modifying the passed in `kw` without calling *start_keyword()* or *end_keyword()* nor visiting child keywords.

visit_message(*msg*)
Implements visiting the message.

Can be overridden to allow modifying the passed in `msg` without calling *start_message()* or *end_message()*.

visit_suite(*suite*)
Implements traversing through the suite and its direct children.

Can be overridden to allow modifying the passed in `suite` without calling *start_suite()* or *end_suite()* nor visiting child suites, tests or keywords (setup and teardown) at all.

robot.result.messagefilter module

class robot.result.messagefilter.**MessageFilter**(*loglevel*)
Bases: *robot.model.visitor.SuiteVisitor*

start_keyword(*keyword*)
Called when keyword starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

end_keyword(*keyword*)
Called when keyword ends. Default implementation does nothing.

end_message(*msg*)
Called when message ends. Default implementation does nothing.

end_suite(*suite*)
Called when suite ends. Default implementation does nothing.

end_test(*test*)
Called when test ends. Default implementation does nothing.

start_message(*msg*)
Called when message starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_suite(*suite*)
Called when suite starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_test(*test*)
Called when test starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

visit_keyword(*kw*)
Implements traversing through the keyword and its child keywords.

Can be overridden to allow modifying the passed in `kw` without calling *`start_keyword()`* or *`end_keyword()`* nor visiting child keywords.

visit_message(*msg*)
Implements visiting the message.

Can be overridden to allow modifying the passed in `msg` without calling *`start_message()`* or *`end_message()`*.

visit_suite(*suite*)
Implements traversing through the suite and its direct children.

Can be overridden to allow modifying the passed in `suite` without calling *`start_suite()`* or *`end_suite()`* nor visiting child suites, tests or keywords (setup and teardown) at all.

visit_test(*test*)
Implements traversing through the test and its keywords.

Can be overridden to allow modifying the passed in `test` without calling *`start_test()`* or *`end_test()`* nor visiting keywords.

robot.result.model module

Module implementing result related model objects.

During test execution these objects are created internally by various runners. At that time they can inspected and modified by listeners.

When results are parsed from XML output files after execution to be able to create logs and reports, these objects are created by the *`ExecutionResult()`* factory method. At that point they can be inspected and modified by pre-Rebot modifiers.

The *ExecutionResult()* factory method can also be used by custom scripts and tools. In such usage it is often easiest to inspect and modify these objects using the *visitor interface*.

class robot.result.model.**Message**(*message=''*, *level='INFO'*, *html=False*, *timestamp=None*, *parent=None*)

Bases: *robot.model.message.Message*

Represents a single log message.

See the base class for documentation of attributes not documented here.

copy(***attributes*)

Return shallow copy of this object.

Parameters **attributes** – Attributes to be set for the returned copy automatically. For example, `test.copy(name='New name')`.

See also *deepcopy()*. The difference between these two is the same as with the standard `copy.copy` and `copy.deepcopy` functions that these methods also use internally.

New in Robot Framework 3.0.1.

deepcopy(***attributes*)

Return deep copy of this object.

Parameters **attributes** – Attributes to be set for the returned copy automatically. For example, `test.deepcopy(name='New name')`.

See also *copy()*. The difference between these two is the same as with the standard `copy.copy` and `copy.deepcopy` functions that these methods also use internally.

New in Robot Framework 3.0.1.

html

html_message

Returns the message content as HTML.

level

message

parent

timestamp

visit(*visitor*)

Visitor interface entry-point.

class robot.result.model.**Keyword**(*kwname=''*, *libname=''*, *doc=''*, *args=()*, *assign=()*, *tags=()*, *timeout=None*, *type='kw'*, *status='FAIL'*, *starttime=None*, *endtime=None*)

Bases: *robot.model.keyword.Keyword*

Represents results of a single keyword.

See the base class for documentation of attributes not documented here.

message_class

alias of *Message*

kwname

Name of the keyword without library or resource name.

libname

Name of the library or resource containing this keyword.

status
Execution status as a string. Typically `PASS` or `FAIL`, but library keywords have status `NOT_RUN` in the dry-ryn mode. See also *passed*.

starttime
Keyword execution start time in format `%Y%m%d %H:%M:%S.%f`.

endtime
Keyword execution end time in format `%Y%m%d %H:%M:%S.%f`.

message
Keyword status message. Used only if suite teardowns fails.

elapsedtime
Total execution time in milliseconds.

name
Keyword name in format `libname.kwname`.

Just `kwname` if *libname* is empty. In practice that is the case only with user keywords in the same file as the executed test case or test suite.

Cannot be set directly. Set *libname* and *kwname* separately instead.

passed
`True` or `False` depending on the *status*.

FOR_ITEM_TYPE = 'foritem'

FOR_LOOP_TYPE = 'for'

KEYWORD_TYPE = 'kw'

SETUP_TYPE = 'setup'

TEARDOWN_TYPE = 'teardown'

args

assign

children
Child *keywords* and *messages* in creation order.

copy(***attributes*)
Return shallow copy of this object.

Parameters **attributes** – Attributes to be set for the returned copy automatically. For example, `test.copy(name='New name')`.

See also *deepcopy()*. The difference between these two is the same as with the standard `copy.copy` and `copy.deepcopy` functions that these methods also use internally.

New in Robot Framework 3.0.1.

deepcopy(***attributes*)
Return deep copy of this object.

Parameters **attributes** – Attributes to be set for the returned copy automatically. For example, `test.deepcopy(name='New name')`.

See also *copy()*. The difference between these two is the same as with the standard `copy.copy` and `copy.deepcopy` functions that these methods also use internally.

New in Robot Framework 3.0.1.

doc

id
Keyword id in format like `s1-t3-k1`.

See *TestSuite.id* for more information.

keyword_class = None

keywords
Child keywords as a *Keywords* object.

messages
Messages as a *Messages* object.

parent
Parent test suite, test case or keyword.

tags
Keyword tags as a *Tags* object.

timeout

type

visit(*visitor*)
Visitor interface entry-point.

class robot.result.model.**TestCase**(*name=''*, *doc=''*, *tags=None*, *timeout=None*, *status='FAIL'*, *message=''*, *starttime=None*, *endtime=None*)
Bases: *robot.model.testcase.TestCase*

Represents results of a single test case.

See the base class for documentation of attributes not documented here.

keyword_class
alias of *Keyword*

status
Status as a string `PASS` or `FAIL`. See also *passed*.

message
Test message. Typically a failure message but can be set also when test passes.

starttime
Test case execution start time in format `%Y%m%d %H:%M:%S.%f`.

endtime
Test case execution end time in format `%Y%m%d %H:%M:%S.%f`.

elapsedtime
Total execution time in milliseconds.

passed
`True/False` depending on the *status*.

critical
`True/False` depending on is the test considered critical.

Criticality is determined based on test's *tags* and *criticality* of the *parent* suite.

copy(***attributes*)
Return shallow copy of this object.

Parameters **attributes** – Attributes to be set for the returned copy automatically. For example, `test.copy(name='New name')`.

See also *deepcopy()*. The difference between these two is the same as with the standard `copy.copy` and `copy.deepcopy` functions that these methods also use internally.

New in Robot Framework 3.0.1.

deepcopy(***attributes*)
Return deep copy of this object.

Parameters **attributes** – Attributes to be set for the returned copy automatically. For example, `test.deepcopy(name='New name')`.

See also *copy()*. The difference between these two is the same as with the standard `copy.copy` and `copy.deepcopy` functions that these methods also use internally.

New in Robot Framework 3.0.1.

doc

id
Test case id in format like `s1-t3`.

See *TestSuite.id* for more information.

keywords
Keywords as a *Keywords* object.

Contains also possible setup and teardown keywords.

longname
Test name prefixed with the long name of the parent suite.

name

parent

tags
Test tags as a *Tags* object.

timeout

visit(*visitor*)
Visitor interface entry-point.

class robot.result.model.**TestSuite**(*name=''*, *doc=''*, *metadata=None*, *source=None*, *message=''*, *starttime=None*, *endtime=None*, *rpa=False*)
Bases: *robot.model.testsuite.TestSuite*

Represents results of a single test suite.

See the base class for documentation of attributes not documented here.

test_class
alias of *TestCase*

keyword_class
alias of *Keyword*

message
Possible suite setup or teardown error message.

starttime
Suite execution start time in format `%Y%m%d %H:%M:%S.%f`.

endtime
Suite execution end time in format `%Y%m%d %H:%M:%S.%f`.

passed
`True` if no critical test has failed, `False` otherwise.

status
`'PASS'` if no critical test has failed, `'FAIL'` otherwise.

statistics
Suite statistics as a *TotalStatistics* object.

Recreated every time this property is accessed, so saving the results to a variable and inspecting it is often a good idea:

```
stats = suite.statistics
print(stats.critical.failed)
print(stats.all.total)
print(stats.message)
```

full_message
Combination of *message* and *stat_message*.

stat_message
String representation of the *statistics*.

elapsedtime
Total execution time in milliseconds.

criticality
Used by tests to determine are they considered critical or not.

Normally configured using `--critical` and `--noncritical` command line options. Can be set programmatically using *set_criticality()* of the root test suite.

set_criticality(*critical_tags=None*, *non_critical_tags=None*)
Sets which tags are considered critical and which non-critical.

Parameters

- **critical_tags** – Tags or patterns considered critical. See the documentation of the `--critical` option for more details.
- **non_critical_tags** – Tags or patterns considered non-critical. See the documentation of the `--noncritical` option for more details.

Tags can be given as lists of strings or, when giving only one, as single strings. This information is used by tests to determine are they considered critical or not.

Criticality can be set only to the root test suite.

remove_keywords(*how*)
Remove keywords based on the given condition.

Parameters **how** – What approach to use when removing keywords. Either `ALL`, `PASSED`, `FOR`, `WUKS`, or `NAME:<pattern>`.

For more information about the possible values see the documentation of the `--removekeywords` command line option.

filter_messages(*log_level='TRACE'*)
Remove log messages below the specified `log_level`.

configure(***options*)
A shortcut to configure a suite using one method call.

Can only be used with the root test suite.

Parameters **options** – Passed to *SuiteConfigurer* that will then set suite attributes, call *filter()*, etc. as needed.

Example:

```
suite.configure(remove_keywords='PASSED',
                critical_tags='smoke',
                doc='Smoke test results.')
```

copy(***attributes*)
Return shallow copy of this object.

Parameters **attributes** – Attributes to be set for the returned copy automatically. For example, `test.copy(name='New name')`.

See also *deepcopy()*. The difference between these two is the same as with the standard `copy.copy` and `copy.deepcopy` functions that these methods also use internally.

New in Robot Framework 3.0.1.

deepcopy(***attributes*)
Return deep copy of this object.

Parameters **attributes** – Attributes to be set for the returned copy automatically. For example, `test.deepcopy(name='New name')`.

See also *copy()*. The difference between these two is the same as with the standard `copy.copy` and `copy.deepcopy` functions that these methods also use internally.

New in Robot Framework 3.0.1.

doc

filter(*included_suites=None*, *included_tests=None*, *included_tags=None*, *excluded_tags=None*)
Select test cases and remove others from this suite.

Parameters have the same semantics as `--suite`, `--test`, `--include`, and `--exclude` command line options. All of them can be given as a list of strings, or when selecting only one, as a single string.

Child suites that contain no tests after filtering are automatically removed.

Example:

```
suite.filter(included_tests=['Test 1', '* Example'],
             included_tags='priority-1')
```

handle_suite_teardown_failures()
Internal usage only.

id
An automatically generated unique id.

The root suite has id `s1`, its child suites have ids `s1-s1`, `s1-s2`, ..., their child suites get ids `s1-s1-s1`, `s1-s1-s2`, ..., `s1-s2-s1`, ..., and so on.

The first test in a suite has an id like `s1-t1`, the second has an id `s1-t2`, and so on. Similarly keywords in suites (setup/teardown) and in tests get ids like `s1-k1`, `s1-t1-k1`, and `s1-s4-t2-k5`.

keywords
Suite setup and teardown as a *Keywords* object.

longname
Suite name prefixed with the long name of the parent suite.

metadata
Free test suite metadata as a dictionary.

name
Test suite name. If not set, constructed from child suite names.

parent

remove_empty_suites()
Removes all child suites not containing any tests, recursively.

rpa

set_tags(*add=None*, *remove=None*, *persist=False*)
Add and/or remove specified tags to the tests in this suite.

Parameters

- **add** – Tags to add as a list or, if adding only one, as a single string.
- **remove** – Tags to remove as a list or as a single string. Can be given as patterns where `*` and `?` work as wildcards.
- **persist** – Add/remove specified tags also to new tests added to this suite in the future.

source

suites
Child suites as a *TestSuites* object.

test_count
Number of the tests in this suite, recursively.

tests
Tests as a *TestCases* object.

visit(*visitor*)
Visitor interface entry-point.

suite_teardown_failed(*message*)
Internal usage only.

robot.result.resultbuilder module

robot.result.resultbuilder.**ExecutionResult**(**sources*, ***options*)
Factory method to constructs *Result* objects.

Parameters

- **sources** – Path(s) to the XML output file(s).
- **options** – Configuration options. Using `merge=True` causes multiple results to be combined so that tests in the latter results replace the ones in the original. Setting `rpa` either to `True` (RPA mode) or `False` (test automation) sets execution mode explicitly. By default it is got from processed output files and conflicting modes cause an error. Other options are passed directly to the *ExecutionResultBuilder* object used internally.

Returns *Result* instance.

Should be imported by external code via the *robot.api* package. See the *robot.result* package for a usage example.

class robot.result.resultbuilder.**ExecutionResultBuilder**(*source*, *include_keywords=True*, *flattened_keywords=None*)

Bases: object

Builds *Result* objects based on output files.

Instead of using this builder directly, it is recommended to use the *ExecutionResult()* factory method.

Parameters

- **source** – Path to the XML output file to build *Result* objects from.
- **include_keywords** – Boolean controlling whether to include keyword information in the result or not. Keywords are not needed when generating only report.
- **flatten_keywords** – List of patterns controlling what keywords to flatten. See the documentation of `--flattenkeywords` option for more details.

build(*result*)

class robot.result.resultbuilder.**RemoveKeywords**

Bases: *robot.model.visitor.SuiteVisitor*

start_suite(*suite*)
Called when suite starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

visit_test(*test*)
Implements traversing through the test and its keywords.

Can be overridden to allow modifying the passed in `test` without calling *start_test()* or *end_test()* nor visiting keywords.

end_keyword(*keyword*)
Called when keyword ends. Default implementation does nothing.

end_message(*msg*)
Called when message ends. Default implementation does nothing.

end_suite(*suite*)
Called when suite ends. Default implementation does nothing.

end_test(*test*)
Called when test ends. Default implementation does nothing.

start_keyword(*keyword*)
Called when keyword starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_message(*msg*)
Called when message starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_test(*test*)
Called when test starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

visit_keyword(*kw*)
Implements traversing through the keyword and its child keywords.

Can be overridden to allow modifying the passed in `kw` without calling *start_keyword()* or *end_keyword()* nor visiting child keywords.

visit_message(*msg*)
Implements visiting the message.

Can be overridden to allow modifying the passed in `msg` without calling *start_message()* or *end_message()*.

visit_suite(*suite*)
Implements traversing through the suite and its direct children.

Can be overridden to allow modifying the passed in `suite` without calling *start_suite()* or *end_suite()* nor visiting child suites, tests or keywords (setup and teardown) at all.

robot.result.suiteteardownfailed module

class robot.result.suiteteardownfailed.**SuiteTeardownFailureHandler**
Bases: *robot.model.visitor.SuiteVisitor*

end_suite(*suite*)
Called when suite ends. Default implementation does nothing.

visit_test(*test*)
Implements traversing through the test and its keywords.

Can be overridden to allow modifying the passed in `test` without calling *start_test()* or *end_test()* nor visiting keywords.

visit_keyword(*keyword*)
Implements traversing through the keyword and its child keywords.

Can be overridden to allow modifying the passed in `kw` without calling *start_keyword()* or *end_keyword()* nor visiting child keywords.

end_keyword(*keyword*)
Called when keyword ends. Default implementation does nothing.

end_message(*msg*)
Called when message ends. Default implementation does nothing.

end_test(*test*)
Called when test ends. Default implementation does nothing.

start_keyword(*keyword*)
Called when keyword starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_message(*msg*)
Called when message starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_suite(*suite*)
Called when suite starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_test(*test*)
Called when test starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

visit_message(*msg*)
Implements visiting the message.

Can be overridden to allow modifying the passed in `msg` without calling *start_message()* or *end_message()*.

visit_suite(*suite*)
Implements traversing through the suite and its direct children.

Can be overridden to allow modifying the passed in `suite` without calling *start_suite()* or *end_suite()* nor visiting child suites, tests or keywords (setup and teardown) at all.

class robot.result.suiteteardownfailed.**SuiteTeardownFailed**(*error*)
Bases: *robot.model.visitor.SuiteVisitor*

visit_test(*test*)
Implements traversing through the test and its keywords.

Can be overridden to allow modifying the passed in `test` without calling *start_test()* or *end_test()* nor visiting keywords.

visit_keyword(*keyword*)
Implements traversing through the keyword and its child keywords.

Can be overridden to allow modifying the passed in `kw` without calling *start_keyword()* or *end_keyword()* nor visiting child keywords.

end_keyword(*keyword*)
Called when keyword ends. Default implementation does nothing.

end_message(*msg*)
Called when message ends. Default implementation does nothing.

end_suite(*suite*)
Called when suite ends. Default implementation does nothing.

end_test(*test*)
Called when test ends. Default implementation does nothing.

start_keyword(*keyword*)
Called when keyword starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_message(*msg*)
Called when message starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_suite(*suite*)
Called when suite starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_test(*test*)
Called when test starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

visit_message(*msg*)
Implements visiting the message.

Can be overridden to allow modifying the passed in `msg` without calling *start_message()* or *end_message()*.

visit_suite(*suite*)
Implements traversing through the suite and its direct children.

Can be overridden to allow modifying the passed in `suite` without calling *start_suite()* or *end_suite()* nor visiting child suites, tests or keywords (setup and teardown) at all.

robot.result.visitor module

Visitors can be used to easily traverse result structures.

This module contains *ResultVisitor* for traversing the whole *Result* object. It extends *SuiteVisitor* that contains visiting logic for the test suite structure.

class robot.result.visitor.**ResultVisitor**
Bases: *robot.model.visitor.SuiteVisitor*

Abstract class to conveniently travel *Result* objects.

A visitor implementation can be given to the `visit()` method of a result object. This will cause the result object to be traversed and the visitor's `visit_x()`, `start_x()`, and `end_x()` methods to be called for each suite, test, keyword and message, as well as for errors, statistics, and other information in the result object. See methods below for a full list of available visitor methods.

See the *result package level* documentation for more information about handling results and a concrete visitor example. For more information about the visitor algorithm see documentation in *robot.model.visitor* module.

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(*suite_stats*)

visit_stat(*stat*)

start_stat(*stat*)

end_stat(*stat*)

visit_errors(*errors*)

start_errors(*errors*)

end_errors(*errors*)

end_keyword(*keyword*)
Called when keyword ends. Default implementation does nothing.

end_message(*msg*)
Called when message ends. Default implementation does nothing.

end_suite(*suite*)
Called when suite ends. Default implementation does nothing.

end_test(*test*)
Called when test ends. Default implementation does nothing.

start_keyword(*keyword*)
Called when keyword starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_message(*msg*)
Called when message starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_suite(*suite*)
Called when suite starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_test(*test*)
Called when test starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

visit_keyword(*kw*)
Implements traversing through the keyword and its child keywords.

Can be overridden to allow modifying the passed in `kw` without calling *`start_keyword()`* or *`end_keyword()`* nor visiting child keywords.

visit_message(*msg*)
Implements visiting the message.

Can be overridden to allow modifying the passed in `msg` without calling *`start_message()`* or *`end_message()`*.

visit_suite(*suite*)
Implements traversing through the suite and its direct children.

Can be overridden to allow modifying the passed in `suite` without calling *`start_suite()`* or *`end_suite()`* nor visiting child suites, tests or keywords (setup and teardown) at all.

visit_test(*test*)
Implements traversing through the test and its keywords.

Can be overridden to allow modifying the passed in `test` without calling *`start_test()`* or *`end_test()`* nor visiting keywords.

robot.result.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*)

get_child_handler(*elem*)

start(*elem*, *result*)

class robot.result.xmlelementhandlers.**RobotHandler**

Bases: robot.result.xmlelementhandlers._Handler

tag = 'robot'

start(*elem*, *result*)

end(*elem*, *result*)

get_child_handler(*elem*)

class robot.result.xmlelementhandlers.**SuiteHandler**

Bases: robot.result.xmlelementhandlers._Handler

tag = 'suite'

start(*elem*, *result*)

end(*elem*, *result*)

get_child_handler(*elem*)

class robot.result.xmlelementhandlers.**RootSuiteHandler**

Bases: *robot.result.xmlelementhandlers.SuiteHandler*

start(*elem*, *result*)

end(*elem*, *result*)

get_child_handler(*elem*)

tag = 'suite'

class robot.result.xmlelementhandlers.**TestCaseHandler**

Bases: robot.result.xmlelementhandlers._Handler

tag = 'test'

start(*elem*, *result*)

end(*elem*, *result*)

get_child_handler(*elem*)

class robot.result.xmlelementhandlers.**KeywordHandler**

Bases: robot.result.xmlelementhandlers._Handler

tag = 'kw'

start(*elem*, *result*)

end(*elem*, *result*)

get_child_handler(*elem*)

class robot.result.xmlelementhandlers.**MessageHandler**
Bases: robot.result.xmlelementhandlers._Handler

tag = 'msg'

end(*elem*, *result*)

get_child_handler(*elem*)

start(*elem*, *result*)

class robot.result.xmlelementhandlers.**KeywordStatusHandler**
Bases: robot.result.xmlelementhandlers._StatusHandler

end(*elem*, *result*)

get_child_handler(*elem*)

start(*elem*, *result*)

tag = 'status'

class robot.result.xmlelementhandlers.**SuiteStatusHandler**
Bases: robot.result.xmlelementhandlers._StatusHandler

end(*elem*, *result*)

get_child_handler(*elem*)

start(*elem*, *result*)

tag = 'status'

class robot.result.xmlelementhandlers.**TestStatusHandler**
Bases: robot.result.xmlelementhandlers._StatusHandler

end(*elem*, *result*)

get_child_handler(*elem*)

start(*elem*, *result*)

tag = 'status'

class robot.result.xmlelementhandlers.**DocHandler**
Bases: robot.result.xmlelementhandlers._Handler

tag = 'doc'

end(*elem*, *result*)

get_child_handler(*elem*)

start(*elem*, *result*)

class robot.result.xmlelementhandlers.**MetadataHandler**
Bases: robot.result.xmlelementhandlers._Handler

tag = 'metadata'

end(*elem*, *result*)

get_child_handler(*elem*)

start(*elem*, *result*)

class robot.result.xmlelementhandlers.**MetadataItemHandler**
Bases: robot.result.xmlelementhandlers._Handler

tag = 'item'

end(*elem*, *result*)

get_child_handler(*elem*)

start(*elem*, *result*)

class robot.result.xmlelementhandlers.**TagsHandler**
Bases: robot.result.xmlelementhandlers._Handler

tag = 'tags'

end(*elem*, *result*)

get_child_handler(*elem*)

start(*elem*, *result*)

class robot.result.xmlelementhandlers.**TagHandler**
Bases: robot.result.xmlelementhandlers._Handler

tag = 'tag'

end(*elem*, *result*)

get_child_handler(*elem*)

start(*elem*, *result*)

class robot.result.xmlelementhandlers.**TimeoutHandler**
Bases: robot.result.xmlelementhandlers._Handler

tag = 'timeout'

end(*elem*, *result*)

get_child_handler(*elem*)

start(*elem*, *result*)

class robot.result.xmlelementhandlers.**AssignHandler**
Bases: robot.result.xmlelementhandlers._Handler

tag = 'assign'

end(*elem*, *result*)

get_child_handler(*elem*)

start(*elem*, *result*)

class robot.result.xmlelementhandlers.**AssignVarHandler**
Bases: robot.result.xmlelementhandlers._Handler

tag = 'var'

end(*elem*, *result*)

get_child_handler(*elem*)

start(*elem*, *result*)

class robot.result.xmlelementhandlers.**ArgumentsHandler**
Bases: robot.result.xmlelementhandlers._Handler

tag = 'arguments'

end(*elem*, *result*)

get_child_handler(*elem*)

start(*elem*, *result*)

class robot.result.xmlelementhandlers.**ArgumentHandler**
Bases: robot.result.xmlelementhandlers._Handler

tag = 'arg'

end(*elem*, *result*)

get_child_handler(*elem*)

start(*elem*, *result*)

class robot.result.xmlelementhandlers.**ErrorsHandler**
Bases: robot.result.xmlelementhandlers._Handler

tag = 'errors'

start(*elem*, *result*)

end(*elem*, *result*)

get_child_handler(*elem*)

class robot.result.xmlelementhandlers.**StatisticsHandler**
Bases: robot.result.xmlelementhandlers._Handler

tag = 'statistics'

get_child_handler(*elem*)

end(*elem*, *result*)

start(*elem*, *result*)

robot.running package

Implements the core test execution logic.

The main public entry points of this package are of the following two classes:

- *TestSuiteBuilder* for creating executable test suites based on existing test case files and directories.
- *TestSuite* for creating an executable test suite structure programmatically.

It is recommended to import both of these classes via the *robot.api* package like in the examples below. Also *TestCase* and *Keyword* classes used internally by the *TestSuite* class are part of the public API. In those rare cases where these classes are needed directly, they can be imported from this package.

Examples

First, let's assume we have the following test suite in file activate_skynet.robot:

```
*** Settings ***
Library    OperatingSystem

*** Test Cases ***
Should Activate Skynet
    [Tags]    smoke
    [Setup]    Set Environment Variable    SKYNET    activated
    Environment Variable Should Be Set    SKYNET
```

We can easily parse and create an executable test suite based on the above file using the *TestSuiteBuilder* class as follows:

```
from robot.api import TestSuiteBuilder

suite = TestSuiteBuilder().build('path/to/activate_skynet.robot')
```

That was easy. Let's next generate the same test suite from scratch using the *TestSuite* class:

```
from robot.api import TestSuite

suite = TestSuite('Activate Skynet')
suite.resource.imports.library('OperatingSystem')
test = suite.tests.create('Should Activate Skynet', tags=['smoke'])
test.keywords.create('Set Environment Variable', args=['SKYNET', 'activated'], type=
↪'setup')
test.keywords.create('Environment Variable Should Be Set', args=['SKYNET'])
```

Not that complicated either, especially considering the flexibility. Notice that the suite created based on the file could also be edited further using the same API.

Now that we have a test suite ready, let's *execute it* and verify that the returned *Result* object contains correct information:

```
result = suite.run(critical='smoke', output='skynet.xml')

assert result.return_code == 0
assert result.suite.name == 'Activate Skynet'
test = result.suite.tests[0]
assert test.name == 'Should Activate Skynet'
assert test.passed and test.critical
stats = result.suite.statistics
assert stats.critical.total == 1 and stats.critical.failed == 0
```

Running the suite generates a normal output XML file, unless it is disabled by using `output=None`. Generating log, report, and xUnit files based on the results is possible using the *ResultWriter* class:

```
from robot.api import ResultWriter

# Report and xUnit files can be generated based on the result object.
ResultWriter(result).write_results(report='skynet.html', log=None)
# Generating log files requires processing the earlier generated output XML.
ResultWriter('skynet.xml').write_results()
```

Subpackages

robot.running.arguments package

Submodules

robot.running.arguments.argumentmapper module

class robot.running.arguments.argumentmapper.**ArgumentMapper**(*argspec*)
Bases: object

map(*positional*, *named*, *replace_defaults=True*)

class robot.running.arguments.argumentmapper.**KeywordCallTemplate**(*argspec*)
Bases: object

fill_positional(*positional*)

fill_named(*named*)

replace_defaults()

class robot.running.arguments.argumentmapper.**DefaultValue**(*value*)
Bases: object

resolve(*variables*)

robot.running.arguments.argumentparser module

robot.running.arguments.argumentparser.**getfullargspec**(*func*)

class robot.running.arguments.argumentparser.**PythonArgumentParser**(*type='Keyword'*)
Bases: robot.running.arguments.argumentparser._ArgumentParser

parse(*handler*, *name=None*)

class robot.running.arguments.argumentparser.**JavaArgumentParser**(*type='Keyword'*)
Bases: robot.running.arguments.argumentparser._ArgumentParser

parse(*signatures*, *name=None*)

class robot.running.arguments.argumentparser.**DynamicArgumentParser**(*type='Keyword'*)
Bases: robot.running.arguments.argumentparser._ArgumentSpecParser

parse(*argspec*, *name=None*)

class robot.running.arguments.argumentparser.**UserKeywordArgumentParser**(*type='Keyword'*)
Bases: robot.running.arguments.argumentparser._ArgumentSpecParser

parse(*argspec*, *name=None*)

robot.running.arguments.argumentresolver module

class robot.running.arguments.argumentresolver.**ArgumentResolver**(*argspec*, *resolve_named=True*, *resolve_variables_until=None*, *dict_to_kwargs=False*)

Bases: object

resolve(*arguments*, *variables=None*)

class robot.running.arguments.argumentresolver.**NamedArgumentResolver**(*argspec*)

Bases: object

resolve(*arguments*, *variables=None*)

class robot.running.arguments.argumentresolver.**NullNamedArgumentResolver**

Bases: object

resolve(*arguments*, *variables=None*)

class robot.running.arguments.argumentresolver.**DictToKwargs**(*argspec*, *enabled=False*)

Bases: object

handle(*positional*, *named*)

class robot.running.arguments.argumentresolver.**VariableReplacer**(*resolve_until=None*)

Bases: object

replace(*positional*, *named*, *variables=None*)

robot.running.arguments.argumentspec module

class robot.running.arguments.argumentspec.**ArgumentSpec**(*name=None*, *type='Keyword'*, *positional=None*, *varargs=None*, *kwonlyargs=None*, *kwargs=None*, *defaults=None*, *types=None*, *supports_named=True*)

Bases: object

types

minargs

maxargs

argument_names

resolve(*arguments*, *variables=None*, *resolve_named=True*, *resolve_variables_until=None*, *dict_to_kwargs=False*)

map(*positional*, *named*, *replace_defaults=True*)

robot.running.arguments.argumentvalidator module

class robot.running.arguments.argumentvalidator.**ArgumentValidator**(*argspec*)
Bases: object

validate(*positional*, *named*, *dryrun=False*)

robot.running.arguments.embedded module

class robot.running.arguments.embedded.**EmbeddedArguments**(*name*)
Bases: object

class robot.running.arguments.embedded.**EmbeddedArgumentParser**
Bases: object

parse(*string*)

robot.running.arguments.javaargumentcoercer module

robot.running.timeouts package

class robot.running.timeouts.**TestTimeout**(*timeout=None*, *message=''*, *variables=None*, *rpa=False*)
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

active

get_message()

replace_variables(*variables*)

run(*runnable*, *args=None*, *kwargs=None*)

start()

time_left()

timed_out()

type = 'Keyword'

Submodules

robot.running.timeouts.ironpython module

robot.running.timeouts.jython module

robot.running.timeouts.posix module

class robot.running.timeouts.posix.**Timeout**(*timeout*, *error*)

Bases: object

execute(*runnable*)

robot.running.timeouts.windows module

class robot.running.timeouts.windows.**Timeout**(*timeout*, *error*)

Bases: object

execute(*runnable*)

Submodules

robot.running.builder module

class robot.running.builder.**TestSuiteBuilder**(*include_suites=None*, *warn_on_skipped='DEPRECATED'*, *extension=None*, *rpa=None*)

Bases: object

Creates executable *TestSuite* objects.

Suites are build based on existing test data on the file system.

See the overall documentation of the *robot.running* package for more information and examples.

Parameters

- **include_suites** – List of suite names to include. If None or an empty list, all suites are included. When executing tests normally, these names are specified using the --suite option.
- **warn_on_skipped** – Deprecated.
- **extension** – Limit parsing test data to only these files. Files are specified as an extension that is handled case-insensitively. Same as --extension on the command line.
- **rpa** – Explicit test execution mode. True for RPA and False for test automation. By default mode is got from test data headers and possible conflicting headers cause an error.

build(**paths*)

Parameters **paths** – Paths to test data files or directories.

Returns *TestSuite* instance.

class robot.running.builder.**ResourceFileBuilder**
Bases: object

build(*path_or_data*, *target=None*)

class robot.running.builder.**StepBuilder**
Bases: object

build_steps(*parent*, *data*, *template=None*, *kw_type='kw'*)

build_step(*parent*, *data*, *template=None*, *kw_type='kw'*)

robot.running.context module

class robot.running.context.**ExecutionContexts**
Bases: object

current

top

namespaces

start_suite(*suite*, *namespace*, *output*, *dry_run=False*)

end_suite()

robot.running.defaults module

class robot.running.defaults.**TestDefaults**(*settings*, *parent=None*)
Bases: object

get_test_values(*test*)

class robot.running.defaults.**TestValues**(*test*, *defaults*)
Bases: object

robot.running.dynamicmethods module

robot.running.dynamicmethods.**no_dynamic_method**(**args*)

class robot.running.dynamicmethods.**GetKeywordNames**(*lib*)
Bases: robot.running.dynamicmethods._DynamicMethod

name

class robot.running.dynamicmethods.**RunKeyword**(*lib*)
Bases: robot.running.dynamicmethods._DynamicMethod

supports_kwargs

name

class robot.running.dynamicmethods.**GetKeywordDocumentation**(*lib*)
Bases: robot.running.dynamicmethods._DynamicMethod

name

class robot.running.dynamicmethods.**GetKeywordArguments**(*lib*)
Bases: robot.running.dynamicmethods._DynamicMethod

name

class robot.running.dynamicmethods.**GetKeywordTypes**(*lib*)
Bases: robot.running.dynamicmethods._DynamicMethod

name

class robot.running.dynamicmethods.**GetKeywordTags**(*lib*)
Bases: robot.running.dynamicmethods._DynamicMethod

name

robot.running.handlers module

robot.running.handlers.**Handler**(*library*, *name*, *method*)

robot.running.handlers.**DynamicHandler**(*library*, *name*, *method*, *doc*, *argspec*, *tags=None*)

robot.running.handlers.**InitHandler**(*library*, *method*, *docgetter=None*)

class robot.running.handlers.**EmbeddedArgumentsHandler**(*name_regexp*, *orig_handler*)
Bases: object

library

matches(*name*)

create_runner(*name*)

robot.running.handlerstore module

class robot.running.handlerstore.**HandlerStore**(*source*, *source_type*)
Bases: object

TEST_LIBRARY_TYPE = 'Test library'

TEST_CASE_FILE_TYPE = 'Test case file'

RESOURCE_FILE_TYPE = 'Resource file'

add(*handler*, *embedded=False*)

create_runner(*name*)

robot.running.importer module

class robot.running.importer.**Importer**
Bases: object

reset()

close_global_library_listeners()

import_library(*name*, *args*, *alias*, *variables*)

import_resource(*path*)

class robot.running.importer.**ImportCache**
Bases: object

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()

robot.running.librarykeywordrunner module

class robot.running.librarykeywordrunner.**LibraryKeywordRunner**(*handler*, *name=None*)
Bases: object

library

libname

longname

run(*kw*, *context*)

dry_run(*kw*, *context*)

class robot.running.librarykeywordrunner.**EmbeddedArgumentsRunner**(*handler*, *name*)
Bases: *robot.running.librarykeywordrunner.LibraryKeywordRunner*

dry_run(*kw*, *context*)

libname

library

longname

run(*kw*, *context*)

class robot.running.librarykeywordrunner.**RunKeywordRunner**(*handler*, *default_dry_run_keywords=False*)
Bases: *robot.running.librarykeywordrunner.LibraryKeywordRunner*

dry_run(*kw*, *context*)

libname

library

longname

run(*kw*, *context*)

robot.running.libraryscopes module

robot.running.libraryscopes.**LibraryScope**(*libcode*, *library*)

class robot.running.libraryscopes.**GlobalScope**(*library*)
Bases: object

is_global = True

start_suite()

end_suite()

start_test()

end_test()

class robot.running.libraryscopes.**TestSuiteScope**(*library*)
Bases: *robot.running.libraryscopes.GlobalScope*

is_global

start_suite()

end_suite()

end_test()

start_test()

class robot.running.libraryscopes.**TestCaseScope**(*library*)
Bases: *robot.running.libraryscopes.TestSuiteScope*

start_test()

end_test()

end_suite()

is_global

start_suite()

robot.running.model module

Module implementing test execution related model objects.

When tests are executed normally, these objects are created based on the test data on the file system by *TestSuiteBuilder*, but external tools can also create an executable test suite model structure directly. Regardless the approach to create it, the model is executed by calling *run()* method of the root test suite. See the *robot.running* package level documentation for more information and examples.

The most important classes defined in this module are *TestSuite*, *TestCase* and *Keyword*. When tests are executed, these objects can be inspected and modified by pre-run modifiers and listeners. The aforementioned objects are considered stable, but other objects in this module may still be changed in the future major releases.

class robot.running.model.**Keyword**(*name=''*, *doc=''*, *args=()*, *assign=()*, *tags=()*, *timeout=None*, *type='kw'*)
Bases: *robot.model.keyword.Keyword*

Represents a single executable keyword.

These keywords never have child keywords or messages. The actual keyword that is executed depends on the context where this model is executed.

See the base class for documentation of attributes not documented here.

message_class = None
Internal usage only.

run(*context*)
Execute the keyword.

Typically called internally by *TestSuite.run()*.

FOR_ITEM_TYPE = 'foritem'

FOR_LOOP_TYPE = 'for'

KEYWORD_TYPE = 'kw'

SETUP_TYPE = 'setup'

TEARDOWN_TYPE = 'teardown'

args

assign

children
Child *keywords* and *messages* in creation order.

copy(***attributes*)
Return shallow copy of this object.

Parameters **attributes** – Attributes to be set for the returned copy automatically. For example, `test.copy(name='New name')`.

See also *deepcopy()*. The difference between these two is the same as with the standard `copy.copy` and `copy.deepcopy` functions that these methods also use internally.

New in Robot Framework 3.0.1.

deepcopy(***attributes*)
Return deep copy of this object.

Parameters **attributes** – Attributes to be set for the returned copy automatically. For example, `test.deepcopy(name='New name')`.

See also *copy()*. The difference between these two is the same as with the standard `copy.copy` and `copy.deepcopy` functions that these methods also use internally.

New in Robot Framework 3.0.1.

doc

id
Keyword id in format like `s1-t3-k1`.

See *TestSuite.id* for more information.

keyword_class = None

keywords
Child keywords as a *Keywords* object.

messages
Messages as a *Messages* object.

name

parent
Parent test suite, test case or keyword.

tags
Keyword tags as a *Tags* object.

timeout

type

visit(*visitor*)
Visitor interface entry-point.

class robot.running.model.**ForLoop**(*variables*, *values*, *flavor*)
Bases: *robot.running.model.Keyword*

Represents a for loop in test data.

Contains keywords in the loop body as child *keywords*.

keyword_class
Internal usage only.

alias of *Keyword*

flavor

variables

values

FOR_ITEM_TYPE = 'foritem'

FOR_LOOP_TYPE = 'for'

KEYWORD_TYPE = 'kw'

SETUP_TYPE = 'setup'

TEARDOWN_TYPE = 'teardown'

args

assign

children
Child *keywords* and *messages* in creation order.

copy(***attributes*)
Return shallow copy of this object.

Parameters **attributes** – Attributes to be set for the returned copy automatically. For example, `test.copy(name='New name')`.

See also *deepcopy()*. The difference between these two is the same as with the standard `copy.copy` and `copy.deepcopy` functions that these methods also use internally.

New in Robot Framework 3.0.1.

deepcopy(***attributes*)
Return deep copy of this object.

Parameters **attributes** – Attributes to be set for the returned copy automatically. For example, `test.deepcopy(name='New name')`.

See also *copy()*. The difference between these two is the same as with the standard `copy.copy` and `copy.deepcopy` functions that these methods also use internally.

New in Robot Framework 3.0.1.

doc

id
Keyword id in format like `s1-t3-k1`.

See *TestSuite.id* for more information.

keywords
Child keywords as a *Keywords* object.

message_class = None

messages
Messages as a *Messages* object.

name

parent
Parent test suite, test case or keyword.

run(*context*)
Execute the keyword.

Typically called internally by *TestSuite.run()*.

tags
Keyword tags as a *Tags* object.

timeout

type

visit(*visitor*)
Visitor interface entry-point.

class robot.running.model.**TestCase**(*name=''*, *doc=''*, *tags=None*, *timeout=None*, *template=None*)
Bases: *robot.model.testcase.TestCase*

Represents a single executable test case.

See the base class for documentation of attributes not documented here.

keyword_class
Internal usage only.

alias of *Keyword*

template
Name of the keyword that has been used as template when building the test. `None` if no is template used.

timeout
Test timeout as a *Timeout* instance or `None`.

This attribute is likely to change in the future.

copy(***attributes*)
Return shallow copy of this object.

Parameters **attributes** – Attributes to be set for the returned copy automatically. For example, `test.copy(name='New name')`.

See also *deepcopy()*. The difference between these two is the same as with the standard `copy.copy` and `copy.deepcopy` functions that these methods also use internally.

New in Robot Framework 3.0.1.

deepcopy(***attributes*)
Return deep copy of this object.

Parameters **attributes** – Attributes to be set for the returned copy automatically. For example, `test.deepcopy(name='New name')`.

See also *copy()*. The difference between these two is the same as with the standard `copy.copy` and `copy.deepcopy` functions that these methods also use internally.

New in Robot Framework 3.0.1.

doc

id
Test case id in format like `s1-t3`.

See *TestSuite.id* for more information.

keywords
Keywords as a *Keywords* object.

Contains also possible setup and teardown keywords.

longname
Test name prefixed with the long name of the parent suite.

name

parent

tags
Test tags as a *Tags* object.

visit(*visitor*)
Visitor interface entry-point.

class robot.running.model.**TestSuite**(*name=''*, *doc=''*, *metadata=None*, *source=None*, *rpa=False*)
Bases: *robot.model.testsuite.TestSuite*

Represents a single executable test suite.

See the base class for documentation of attributes not documented here.

test_class
Internal usage only.

alias of *TestCase*

keyword_class
Internal usage only.

alias of *Keyword*

resource
ResourceFile instance containing imports, variables and keywords the suite owns. When data is parsed from the file system, this data comes from the same test case file that creates the suite.

configure(*randomize_suites=False*, *randomize_tests=False*, *randomize_seed=None*, ***options*)
A shortcut to configure a suite using one method call.

Can only be used with the root test suite.

Parameters

- **randomize_xxx** – Passed to *randomize()*.
- **options** – Passed to *SuiteConfigurer* that will then set suite attributes, call *filter()*, etc. as needed.

Example:

```
suite.configure(included_tags=['smoke'],
                doc='Smoke test results.')
```

randomize(*suites=True*, *tests=True*, *seed=None*)
Randomizes the order of suites and/or tests, recursively.

Parameters

- **suites** – Boolean controlling should suites be randomized.
- **tests** – Boolean controlling should tests be randomized.
- **seed** – Random seed. Can be given if previous random order needs to be re-created. Seed value is always shown in logs and reports.

run(*settings=None*, ***options*)
Executes the suite based based the given `settings` or `options`.

Parameters

- **settings** – *RobotSettings* object to configure test execution.
- **options** – Used to construct new *RobotSettings* object if `settings` are not given.

Returns *Result* object with information about executed suites and tests.

If `options` are used, their names are the same as long command line options except without hyphens. Some options are ignored (see below), but otherwise they have the same semantics as on the command line. Options that can be given on the command line multiple times can be passed as lists like `variable=['VAR1:value1', 'VAR2:value2']`. If such an option is used only once, it can be given also as a single string like `variable='VAR:value'`.

Additionally listener option allows passing object directly instead of listener name, e.g. `run('tests.robot', listener=Listener())`.

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

Only options related to the actual test execution have an effect. For example, options related to selecting or modifying test cases or suites (e.g. `--include`, `--name`, `--prerunmodifier`) or creating logs and reports are silently ignored. The output XML generated as part of the execution can be configured, though. This includes disabling it with `output=None`.

Example:

```
stdout = StringIO()
result = suite.run(variable='EXAMPLE:value',
                   critical='regression',
                   output='example.xml',
                   exitonfailure=True,
                   stdout=stdout)
print result.return_code
```

To save memory, the returned *Result* object does not have any information about the executed keywords. If that information is needed, the created output XML file needs to be read using the *ExecutionResult* factory method.

See the *package level* documentation for more examples, including how to construct executable test suites and how to create logs and reports based on the execution results.

See the *robot.run* function for a higher-level API for executing tests in files or directories.

copy(***attributes*)
Return shallow copy of this object.

Parameters **attributes** – Attributes to be set for the returned copy automatically. For example, `test.copy(name='New name')`.

See also *deepcopy()*. The difference between these two is the same as with the standard `copy.copy` and `copy.deepcopy` functions that these methods also use internally.

New in Robot Framework 3.0.1.

deepcopy(***attributes*)
Return deep copy of this object.

Parameters **attributes** – Attributes to be set for the returned copy automatically. For example, `test.deepcopy(name='New name')`.

See also *copy()*. The difference between these two is the same as with the standard `copy.copy` and `copy.deepcopy` functions that these methods also use internally.

New in Robot Framework 3.0.1.

doc

filter(*included_suites=None*, *included_tests=None*, *included_tags=None*, *excluded_tags=None*)
Select test cases and remove others from this suite.

Parameters have the same semantics as `--suite`, `--test`, `--include`, and `--exclude` command line options. All of them can be given as a list of strings, or when selecting only one, as a single string.

Child suites that contain no tests after filtering are automatically removed.

Example:

```
suite.filter(included_tests=['Test 1', '* Example'],
             included_tags='priority-1')
```

id
An automatically generated unique id.

The root suite has id `s1`, its child suites have ids `s1-s1`, `s1-s2`, ..., their child suites get ids `s1-s1-s1`, `s1-s1-s2`, ..., `s1-s2-s1`, ..., and so on.

The first test in a suite has an id like `s1-t1`, the second has an id `s1-t2`, and so on. Similarly keywords in suites (setup/teardown) and in tests get ids like `s1-k1`, `s1-t1-k1`, and `s1-s4-t2-k5`.

keywords
Suite setup and teardown as a *Keywords* object.

longname
Suite name prefixed with the long name of the parent suite.

metadata
Free test suite metadata as a dictionary.

name
Test suite name. If not set, constructed from child suite names.

parent

remove_empty_suites()
Removes all child suites not containing any tests, recursively.

rpa

set_tags(*add=None*, *remove=None*, *persist=False*)
Add and/or remove specified tags to the tests in this suite.

Parameters

- **add** – Tags to add as a list or, if adding only one, as a single string.
- **remove** – Tags to remove as a list or as a single string. Can be given as patterns where * and ? work as wildcards.
- **persist** – Add/remove specified tags also to new tests added to this suite in the future.

source

suites
Child suites as a *TestSuites* object.

test_count
Number of the tests in this suite, recursively.

tests
Tests as a *TestCases* object.

visit(*visitor*)
Visitor interface entry-point.

class robot.running.model.**Variable**(*name*, *value*, *source=None*)
Bases: object

report_invalid_syntax(*message*, *level='ERROR'*)

class robot.running.model.**Timeout**(*value*, *message=None*)
Bases: object

class robot.running.model.**ResourceFile**(*doc=''*, *source=None*)
Bases: object

imports

keywords

variables

class robot.running.model.**UserKeyword**(*name*, *args=()*, *doc=''*, *tags=()*, *return_=None*, *timeout=None*)
Bases: object

keywords

timeout
Keyword timeout as a *Timeout* instance or None.

tags

robot.running.namespace module

class robot.running.namespace.**Namespace**(*variables*, *suite*, *resource*)
Bases: object

libraries

handle_imports()

import_resource(*name*, *overwrite=True*)

import_variables(*name*, *args*, *overwrite=False*)

import_library(*name*, *args=None*, *alias=None*, *notify=True*)

set_search_order(*new_order*)

start_test()

end_test()

start_suite()

end_suite(*suite*)

start_user_keyword()

end_user_keyword()

get_library_instance(*libname*)

get_library_instances()

reload_library(*libname_or_instance*)

get_runner(*name*)

class robot.running.namespace.**KeywordStore**(*resource*)
Bases: object

get_library(*name_or_instance*)

get_runner(*name*)

class robot.running.namespace.**KeywordRecommendationFinder**(*user_keywords*, *libraries*, *resources*)
Bases: object

recommend_similar_keywords(*name*)
Return keyword names similar to *name*.

static format_recommendations(*msg*, *recommendations*)

robot.running.outputcapture module

class robot.running.outputcapture.**OutputCapturer**(*library_import=False*)
Bases: object

class robot.running.outputcapture.**PythonCapturer**(*stdout=True*)
Bases: object

release()

class robot.running.outputcapture.**JavaCapturer**(*stdout=True*)
Bases: object

release()

robot.running.randomizer module

class robot.running.randomizer.**Randomizer**(*randomize_suites=True*, *randomize_tests=True*, *seed=None*)
Bases: *robot.model.visitor.SuiteVisitor*

start_suite(*suite*)
Called when suite starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

visit_test(*test*)
Implements traversing through the test and its keywords.

Can be overridden to allow modifying the passed in `test` without calling *start_test()* or *end_test()* nor visiting keywords.

visit_keyword(*kw*)
Implements traversing through the keyword and its child keywords.

Can be overridden to allow modifying the passed in `kw` without calling *start_keyword()* or *end_keyword()* nor visiting child keywords.

end_keyword(*keyword*)
Called when keyword ends. Default implementation does nothing.

end_message(*msg*)
Called when message ends. Default implementation does nothing.

end_suite(*suite*)
Called when suite ends. Default implementation does nothing.

end_test(*test*)
Called when test ends. Default implementation does nothing.

start_keyword(*keyword*)
Called when keyword starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_message(*msg*)
Called when message starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_test(*test*)
Called when test starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

visit_message(*msg*)
Implements visiting the message.

Can be overridden to allow modifying the passed in `msg` without calling *start_message()* or *end_message()*.

visit_suite(*suite*)
Implements traversing through the suite and its direct children.

Can be overridden to allow modifying the passed in `suite` without calling *start_suite()* or *end_suite()* nor visiting child suites, tests or keywords (setup and teardown) at all.

robot.running.runkwregister module

robot.running.runner module

class robot.running.runner.**Runner**(*output*, *settings*)
Bases: *robot.model.visitor.SuiteVisitor*

start_suite(*suite*)
Called when suite starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

end_suite(*suite*)
Called when suite ends. Default implementation does nothing.

visit_test(*test*)
Implements traversing through the test and its keywords.

Can be overridden to allow modifying the passed in `test` without calling *start_test()* or *end_test()* nor visiting keywords.

end_keyword(*keyword*)
Called when keyword ends. Default implementation does nothing.

end_message(*msg*)
Called when message ends. Default implementation does nothing.

end_test(*test*)
Called when test ends. Default implementation does nothing.

start_keyword(*keyword*)
Called when keyword starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_message(*msg*)
Called when message starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_test(*test*)
Called when test starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

visit_keyword(*kw*)
Implements traversing through the keyword and its child keywords.

Can be overridden to allow modifying the passed in `kw` without calling *start_keyword()* or *end_keyword()* nor visiting child keywords.

visit_message(*msg*)
Implements visiting the message.

Can be overridden to allow modifying the passed in `msg` without calling *start_message()* or *end_message()*.

visit_suite(*suite*)
Implements traversing through the suite and its direct children.

Can be overridden to allow modifying the passed in `suite` without calling *start_suite()* or *end_suite()* nor visiting child suites, tests or keywords (setup and teardown) at all.

class robot.running.runner.**ModelCombiner**(*data*, *result*, ***priority*)
Bases: `object`

robot.running.signalhandler module

robot.running.status module

class robot.running.status.**Failure**
Bases: object

class robot.running.status.**Exit**(*failure_mode=False*, *error_mode=False*, *skip_teardown_mode=False*)
Bases: object

failure_occurred(*failure=None*, *critical=False*)

error_occurred()

teardown_allowed

class robot.running.status.**SuiteStatus**(*parent=None*, *exit_on_failure_mode=False*, *exit_on_error_mode=False*, *skip_teardown_on_exit_mode=False*)
Bases: robot.running.status._ExecutionStatus

critical_failure_occurred()

error_occurred()

failures

message

setup_executed(*failure=None*)

status

teardown_allowed

teardown_executed(*failure=None*)

class robot.running.status.**TestStatus**(*parent*, *critical*)
Bases: robot.running.status._ExecutionStatus

test_failed(*failure*)

critical_failure_occurred()

error_occurred()

failures

message

setup_executed(*failure=None*)

status

teardown_allowed

teardown_executed(*failure=None*)

class robot.running.status.**TestMessage**(*status*)
Bases: robot.running.status._Message

setup_message = 'Setup failed:\n%s'

teardown_message = 'Teardown failed:\n%s'

also_teardown_message = '%s\n\nAlso teardown failed:\n%s'

exit_on_fatal_message = 'Test execution stopped due to a fatal error.'

exit_on_failure_message = 'Critical failure occurred and exit-on-failure mode is in us

exit_on_error_message = 'Error occurred and exit-on-error mode is in use.'

message

class robot.running.status.**SuiteMessage**(*status*)
Bases: robot.running.status._Message

setup_message = 'Suite setup failed:\n%s'

teardown_message = 'Suite teardown failed:\n%s'

also_teardown_message = '%s\n\nAlso suite teardown failed:\n%s'

message

class robot.running.status.**ParentMessage**(*status*)
Bases: *robot.running.status.SuiteMessage*

setup_message = 'Parent suite setup failed:\n%s'

teardown_message = 'Parent suite teardown failed:\n%s'

also_teardown_message = '%s\n\nAlso parent suite teardown failed:\n%s'

message

robot.running.statusreporter module

class robot.running.statusreporter.**StatusReporter**(*context*, *result*, *dry_run_lib_kw=False*)
Bases: object

robot.running.steprunner module

class robot.running.steprunner.**StepRunner**(*context*, *templated=False*)
Bases: object

run_steps(*steps*)

run_step(*step*, *name=None*)

robot.running.steprunner.**ForRunner**(*context*, *templated=False*, *flavor='IN'*)

class robot.running.steprunner.**ForInRunner**(*context*, *templated=False*)
Bases: object

run(*data*, *name=None*)

class robot.running.steprunner.**ForInRangeRunner**(*context*, *templated=False*)
Bases: *robot.running.steprunner.ForInRunner*

run(*data*, *name=None*)

class robot.running.steprunner.**ForInZipRunner**(*context*, *templated=False*)
Bases: *robot.running.steprunner.ForInRunner*

run(*data*, *name=None*)

class robot.running.steprunner.**ForInEnumerateRunner**(*context*, *templated=False*)
Bases: *robot.running.steprunner.ForInRunner*

run(*data*, *name=None*)

class robot.running.steprunner.**InvalidForRunner**(*context*, *flavor*)
Bases: *robot.running.steprunner.ForInRunner*

Used to send an error from ForRunner() if it sees an unexpected error.

We can't simply throw a DataError from ForRunner() because that happens outside the "with StatusReporter(...)" blocks.

run(*data*, *name=None*)

robot.running.testlibraries module

robot.running.testlibraries.**TestLibrary**(*name*, *args=None*, *variables=None*, *create_handlers=True*)

robot.running.usererrorhandler module

class robot.running.usererrorhandler.**UserErrorHandler**(*error*, *name*, *libname=None*)
Bases: object

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.

Parameters

- **error** (robot.errors.DataError) – Occurred error.
- **name** (*str*) – Name of the affected keyword.
- **libname** (*str*) – Name of the affected library or resource.

longname

doc

shortdoc

create_runner(*name*)

run(*kw*, *context*)

dry_run(*kw*, *context*)

robot.running.userkeyword module

class robot.running.userkeyword.**UserLibrary**(*source*, *source_type='Resource file'*)
Bases: object

TEST_CASE_FILE_TYPE = 'Test case file'

RESOURCE_FILE_TYPE = 'Resource file'

class robot.running.userkeyword.**UserKeywordHandler**(*keyword*, *libname*)
Bases: object

longname

shortdoc

create_runner(*name*)

class robot.running.userkeyword.**EmbeddedArgumentsHandler**(*keyword*, *libname*, *embedded*)

Bases: *robot.running.userkeyword.UserKeywordHandler*

matches(*name*)

create_runner(*name*)

longname

shortdoc

robot.running.userkeywordrunner module

class robot.running.userkeywordrunner.**UserKeywordRunner**(*handler*, *name=None*)

Bases: object

longname

libname

arguments

run(*kw*, *context*)

dry_run(*kw*, *context*)

class robot.running.userkeywordrunner.**EmbeddedArgumentsRunner**(*handler*, *name*)

Bases: *robot.running.userkeywordrunner.UserKeywordRunner*

arguments

dry_run(*kw*, *context*)

libname

longname

run(*kw*, *context*)

robot.utils package

Various generic utility functions and classes.

Utilities are mainly for internal usage, but external libraries and tools may find some of them useful. Utilities are generally stable, but absolute backwards compatibility between major versions is not guaranteed.

All utilities are exposed via the *robot.utils* package, and should be used either like:

```
from robot import utils

assert utils.Matcher('H?llo').match('Hillo')
```

or:

```
from robot.utils import Matcher

assert Matcher('H?llo').match('Hillo')
```

Submodules

robot.utils.application module

class robot.utils.application.**Application**(*usage*, *name=None*, *version=None*, *arg_limits=None*, *env_options=None*, *logger=None*, ***auto_options*)

Bases: object

main(*arguments*, ***options*)

validate(*options*, *arguments*)

execute_cli(*cli_arguments*, *exit=True*)

console(*msg*)

parse_arguments(*cli_args*)

Public interface for parsing command line arguments.

Parameters **cli_args** – Command line arguments as a list

Returns options (dict), arguments (list)

Raises *Information* when –help or –version used

Raises *DataError* when parsing fails

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

class robot.utils.application.**DefaultLogger**

Bases: object

info(*message*)

error(*message*)

close()

robot.utils.argumentparser module

robot.utils.argumentparser.**cmdline2list**(*args*, *escaping=False*)

class robot.utils.argumentparser.**ArgumentParser**(*usage*, *name=None*, *version=None*, *arg_limits=None*, *validator=None*, *env_options=None*, *auto_help=True*, *auto_version=True*, *auto_escape=True*, *auto_pythonpath=True*, *auto_argumentfile=True*)

Bases: object

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*)

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*)

robot.utils.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:

```
import unittest
from robot.utils.asserts import assert_equal

class MyTests(unittest.TestCase):

    def test_old_style(self):
        self.assertEqual(1, 2, 'my msg')

    def test_new_style(self):
        assert_equal(1, 2, 'my msg')
```

Example output:

```
FF
======================================================================
FAIL: test_old_style (example.MyTests)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "example.py", line 7, in test_old_style
    self.assertEqual(1, 2, 'my msg')
AssertionError: my msg

======================================================================
FAIL: test_new_style (example.MyTests)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "example.py", line 10, in test_new_style
    assert_equal(1, 2, 'my msg')
  File "/path/to/robot/utils/asserts.py", line 181, in assert_equal
    _report_inequality_failure(first, second, msg, values, '!=')
  File "/path/to/robot/utils/asserts.py", line 229, in _report_inequality_failure
    raise AssertionError(msg)
AssertionError: my msg: 1 != 2
```

(continues on next page)

(continued from previous page)

```
----------------------------------------------------------------------
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.**assert_false**(*expr*, *msg=None*)
Fail the test if the expression is True.

robot.utils.asserts.**assert_true**(*expr*, *msg=None*)
Fail the test unless the expression is True.

robot.utils.asserts.**assert_not_none**(*obj*, *msg=None*, *values=True*)
Fail the test if given object is None.

robot.utils.asserts.**assert_none**(*obj*, *msg=None*, *values=True*)
Fail the test if given object is not None.

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_equal**(*first*, *second*, *msg=None*, *values=True*, *formatter=None*)
Fail if given objects are unequal as determined by the '==' operator.

robot.utils.asserts.**assert_not_equal**(*first*, *second*, *msg=None*, *values=True*, *formatter=None*)
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.

inequality 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.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.

[1] https://github.com/robotframework/robotframework/issues/604 [2] https://github.com/robotframework/robotframework/issues/1096

`robot.utils.charwidth.`**`get_char_width`**(*char*)

robot.utils.compat module

`robot.utils.compat.`**`py2to3`**(*cls*)

`robot.utils.compat.`**`with_metaclass`**(*meta*, **bases*)
Create a base class with a metaclass.

`robot.utils.compat.`**`isatty`**(*stream*)

robot.utils.compress module

`robot.utils.compress.`**`compress_text`**(*text*)

robot.utils.connectioncache module

class `robot.utils.connectioncache.`**`ConnectionCache`**(*no_current_msg='No open connection.'*)

Bases: `object`

Cache for test libs to use with concurrent connections, processes, etc.

The cache stores the registered connections (or other objects) and allows switching between them using generated indices or user given aliases. This is useful with any test library where there's need for multiple concurrent connections, processes, etc.

This class can, and is, used also outside the core framework by SSHLibrary, Selenium(2)Library, etc. Backwards compatibility is thus important when doing changes.

`current = None`
Current active connection.

`current_index`

`register`(*connection*, *alias=None*)
Registers given connection with optional alias and returns its index.

Given connection is set to be the *`current`* connection.

If alias is given, it must be a string. Aliases are case and space insensitive.

The index of the first connection after initialization, and after *`close_all()`* or *`empty_cache()`*, is 1, second is 2, etc.

`switch`(*alias_or_index*)
Switches to the connection specified by the given alias or index.

Updates *`current`* and also returns its new value.

Alias is whatever was given to *register()* method and indices are returned by it. Index can be given either as an integer or as a string that can be converted to an integer. Raises an error if no connection with the given index or alias found.

get_connection(*alias_or_index=None*)
Get the connection specified by the given alias or index..

If `alias_or_index` is `None`, returns the current connection if it is active, or raises an error if it is not.

Alias is whatever was given to *register()* method and indices are returned by it. Index can be given either as an integer or as a string that can be converted to an integer. Raises an error if no connection with the 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 connection cache.

Indexes of the new connections starts from 1 after this.

resolve_alias_or_index(*alias_or_index*)

class robot.utils.connectioncache.**NoConnection**(*message*)
Bases: `object`

raise_error()

robot.utils.dotdict module

class robot.utils.dotdict.**DotDict**(**args*, ***kwds*)
Bases: `collections.OrderedDict`

clear() → None. Remove all items from od.

copy() → a shallow copy of od

classmethod fromkeys(*S*[, *v*]) → New ordered dictionary with keys from S.
If not specified, the value defaults to None.

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

has_key(*k*) → True if D has a key k, else False

items() → list of (key, value) pairs in od

iteritems()
od.iteritems -> an iterator over the (key, value) pairs in od

iterkeys() → an iterator over the keys in od

itervalues()
od.itervalues -> an iterator over the values in od

keys() → list of keys in od

pop(*k*[, *d*]) → v, remove specified key and return the corresponding
value. If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), return and remove a (key, value) pair.
Pairs are returned in LIFO order if last is true or FIFO order if false.

setdefault(*k*[, *d*]) → od.get(k,d), also set od[k]=d if k not in od

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

values() → list of values in od

viewitems() → a set-like object providing a view on od's items

viewkeys() → a set-like object providing a view on od's keys

viewvalues() → an object providing a view on od's values

robot.utils.encoding module

robot.utils.encoding.**console_decode**(*string*, *encoding='UTF-8'*, *force=False*)
Decodes bytes from console encoding to Unicode.

By default uses the system console encoding, but that can be configured using the *encoding* argument. In addition to the normal encodings, it is possible to use case-insensitive values *CONSOLE* and *SYSTEM* to use the system console and system encoding, respectively.

By default returns Unicode strings as-is. The *force* argument can be used on IronPython where all strings are *unicode* and caller knows decoding is needed.

robot.utils.encoding.**console_encode**(*string*, *errors='replace'*, *stream=<open file '<stdout>', mode 'w'>*)
Encodes Unicode to bytes in console or system encoding.

Determines the encoding to use based on the given stream and system configuration. On Python 3 and IronPython returns Unicode, otherwise returns bytes.

robot.utils.encoding.**system_decode**(*string*)
Decodes bytes from system (e.g. cli args or env vars) to Unicode.

robot.utils.encoding.**system_encode**(*string*, *errors='replace'*)
Encodes Unicode to system encoding (e.g. cli args and env vars).

Non-Unicode values are first converted to Unicode.

robot.utils.encodingsniffer module

robot.utils.encodingsniffer.**get_system_encoding**()

robot.utils.encodingsniffer.**get_console_encoding**()

robot.utils.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**(*exclude_robot_traces=True*)
Returns error message and details of the last occurred exception.

robot.utils.error.**ErrorDetails**(*exc_info=None*, *exclude_robot_traces=True*)
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*, *exclude_robot_traces=True*)
Bases: robot.utils.error._ErrorDetails

message

traceback

class robot.utils.error.**JavaErrorDetails**(*exc_type*, *exc_value*, *exc_traceback*, *exclude_robot_traces=True*)
Bases: robot.utils.error._ErrorDetails

message

traceback

robot.utils.escaping module

robot.utils.escaping.**escape**(*item*)

robot.utils.escaping.**unescape**(*item*)

class robot.utils.escaping.**Unescaper**
Bases: object

unescape(*string*)

class robot.utils.escaping.**EscapeFinder**(*string*)
Bases: object

robot.utils.escaping.**split_from_equals**(*string*)

robot.utils.etreewrapper module

class robot.utils.etreewrapper.**ETSource**(*source*)
Bases: object

robot.utils.frange module

robot.utils.frange.**frange**(**args*)
Like range() but accepts float arguments.

robot.utils.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

match()
match(string[, pos[, endpos]]) –> match object or None. Matches zero or more characters at the beginning of the string

format_line(*line*)

add(*line*)

end()

format(*lines*)

handles(*line*)

class robot.utils.htmlformatters.**HeaderFormatter**
Bases: robot.utils.htmlformatters._SingleLineFormatter

match()
match(string[, pos[, endpos]]) –> match object or None. Matches zero or more characters at the beginning of the string

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*)

robot.utils.importer module

robot.utils.importer.**invalidate_import_caches**()

class robot.utils.importer.**Importer**(*type=None*, *logger=None*)
Bases: object

import_class_or_module(*name*, *instantiate_with_args=None*, *return_source=False*)
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 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 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*)

robot.utils.markuputils module

robot.utils.markuputils.**html_escape**(*text*, *linkify=True*)

robot.utils.markuputils.**xml_escape**(*text*)

robot.utils.markuputils.**html_format**(*text*)

robot.utils.markuputils.**attribute_escape**(*attr*)

robot.utils.markupwriters module

class robot.utils.markupwriters.**HtmlWriter**(*output*, *write_empty=True*)
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.
- **write_empty** – Whether to write empty elements and attributes.

close()
Closes the underlying output file.

content(*content=None*, *escape=True*, *newline=False*)

element(*name*, *content=None*, *attrs=None*, *escape=True*, *newline=True*, *replace_newlines=False*)

end(*name*, *newline=True*)

start(*name*, *attrs=None*, *newline=True*)

class robot.utils.markupwriters.**XmlWriter**(*output*, *write_empty=True*)
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.
- **write_empty** – Whether to write empty elements and attributes.

close()
Closes the underlying output file.

content(*content=None*, *escape=True*, *newline=False*)

element(*name*, *content=None*, *attrs=None*, *escape=True*, *newline=True*, *replace_newlines=False*)

end(*name*, *newline=True*)

start(*name*, *attrs=None*, *newline=True*)

class robot.utils.markupwriters.**NullMarkupWriter**(***kwargs*)
Bases: object

Null implementation of the _MarkupWriter interface.

start(***kwargs*)

content(***kwargs*)

element(***kwargs*)

end(***kwargs*)

close(***kwargs*)

robot.utils.match module

robot.utils.match.**eq**(*str1*, *str2*, *ignore=()*, *caseless=True*, *spaceless=True*)

class robot.utils.match.**Matcher**(*pattern*, *ignore=()*, *caseless=True*, *spaceless=True*, *regexp=False*)
Bases: object

match(*string*)

match_any(*strings*)

class robot.utils.match.**MultiMatcher**(*patterns=None*, *ignore=()*, *caseless=True*, *spaceless=True*, *match_if_no_patterns=False*, *regexp=False*)
Bases: object

match(*string*)

match_any(*strings*)

robot.utils.misc module

robot.utils.misc.**roundup**(*number*, *ndigits=0*, *return_type=None*)
Rounds number to the given number of digits.

Numbers equally close to a certain precision are always rounded away from zero. By default return value is float when ndigits is positive and int otherwise, but that can be controlled with return_type.

With the built-in round() rounding equally close numbers as well as the return type depends on the Python version.

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' '' -> ''

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.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*)

class robot.utils.normalizing.**NormalizedDict**(*initial=None*, *ignore=()*, *caseless=True*, *spaceless=True*)
Bases: `_abcoll.MutableMapping`

Custom dictionary implementation automatically normalizing keys.

Initialized 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 the *`normalize()`* function.

copy()

clear() → None. Remove all items from D.

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

items() → list of D's (key, value) pairs, as 2-tuples

iteritems() → an iterator over the (key, value) items of D

iterkeys() → an iterator over the keys of D

itervalues() → an iterator over the values of D

keys() → list of D's keys

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

values() → list of D's values

robot.utils.ordereddict module

robot.utils.platform module

robot.utils.recommendations module

class robot.utils.recommendations.**RecommendationFinder**(*normalizer=None*)
Bases: object

find_recommendations(*name*, *candidates*, *max_matches=10*)
Return a list of close matches to *name* from *candidates*.

static format_recommendations(*msg*, *recommendations*)
Add recommendations to the given message.

The recommendation string looks like: <msg> Did you mean: <recommendations[0]> <recommendations[1]> <recommendations[2]>

robot.utils.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**(*upper=False*)

robot.utils.robotinspect module

robot.utils.robotinspect.**is_java_init**(*init*)

robot.utils.robotinspect.**is_java_method**(*method*)

robot.utils.robotio module

robot.utils.robotio.**file_writer**(*path=None*, *encoding='UTF-8'*, *newline=None*)

robot.utils.robotio.**binary_file_writer**(*path=None*)

robot.utils.robotpath module

robot.utils.robotpath.**path_to_url**(*path*)

robot.utils.robotpath.**normpath**(*path*, *case_normalize=False*)
Replacement for os.path.normpath with some enhancements.

1. Convert non-Unicode paths to Unicode using the file system encoding.
2. NFC normalize Unicode paths (affects mainly OSX).
3. Optionally lower-case paths on case-insensitive file systems. That includes Windows and also OSX in default configuration.
4. Turn `c:` into `c:\` on Windows instead of keeping it as `c:`.

robot.utils.robotpath.**abspath**(*path*, *case_normalize=False*)
Replacement for os.path.abspath with some enhancements and bug fixes.

1. Non-Unicode paths are converted to Unicode using file system encoding.
2. Optionally lower-case paths on case-insensitive file systems. That includes Windows and also OSX in default configuration.
3. Turn `c:` into `c:\` on Windows instead of `c:\current\path`.

robot.utils.robotpath.**get_link_path**(*target*, *base*)
Returns a relative path to `target` from `base`.

If `base` is an existing file, then its parent directory is considered to be the base. Otherwise `base` is assumed to be a directory.

The returned path is URL encoded. On Windows returns an absolute path with `file:` prefix if the target is on a different drive.

robot.utils.robotpath.**find_file**(*path*, *basedir='.'*, *file_type=None*)

robot.utils.robottime module

robot.utils.robottime.**timestr_to_secs**(*timestr*, *round_to=3*)
Parses time like '1h 10s', '01:00:10' or '42' and returns seconds.

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 maximun 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*)
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 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.

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_elapsed_time**(*start_time*, *end_time*)
Returns the time between given timestamps in milliseconds.

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.

class robot.utils.robottime.**TimestampCache**
Bases: object

get_timestamp(*daysep=''*, *daytimesep=' '*, *timesep=':'*, *millissep='.'*)

robot.utils.robottypes module

robot.utils.robottypes.**is_truthy**(*item*)
Returns *True* or *False* depending is the item considered true or not.

Validation rules:

- If the value is a string, it is considered false if it is *'FALSE'*, *'NO'*, *'OFF'*, *'0'*, *'NONE'* or *''*, case-insensitively. Considering *'NONE'* false is new in RF 3.0.3 and considering *'OFF'* and *'0'* false is new in RF 3.1.
- Other strings are considered true.
- Other values are handled by using the standard *bool()* function.

Designed to be used also by external test libraries that want to handle Boolean values similarly as Robot Framework itself. See also *is_falsy()*.

robot.utils.robottypes.**is_falsy**(*item*)
Opposite of *is_truthy()*.

robot.utils.robottypes2 module

robot.utils.robottypes2.**is_integer**(*item*)

robot.utils.robottypes2.**is_number**(*item*)

robot.utils.robottypes2.**is_bytes**(*item*)

robot.utils.robottypes2.**is_string**(*item*)

robot.utils.robottypes2.**is_unicode**(*item*)

robot.utils.robottypes2.**is_list_like**(*item*)

robot.utils.robottypes2.**is_dict_like**(*item*)

robot.utils.robottypes2.**type_name**(*item*)

robot.utils.robottypes3 module

robot.utils.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

robot.utils.sortable module

class robot.utils.sortable.**Sortable**
Bases: object

Base class for sorting based self._sort_key

robot.utils.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*)

robot.utils.text.**split_args_from_name_or_path**(*name*)

robot.utils.text.**split_tags_from_doc**(*doc*)

robot.utils.text.**getdoc**(*item*)

robot.utils.text.**getshortdoc**(*doc_or_item*, *linesep='\n'*)

robot.utils.unic module

robot.utils.unic.**unic**(*item*)

robot.utils.unic.**prepr**(*item*, *width=80*)

class robot.utils.unic.**PrettyRepr**(*indent=1*, *width=80*, *depth=None*, *stream=None*)
Bases: pprint.PrettyPrinter

Handle pretty printing operations onto a stream using a set of configured parameters.

indent Number of spaces to indent for each level of nesting.

width Attempted maximum number of columns in the output.

depth The maximum depth to print out nested structures.

stream The desired output stream. If omitted (or false), the standard output stream available at construction will be used.

format(*object*, *context*, *maxlevels*, *level*)

isreadable(*object*)

isrecursive(*object*)

pformat(*object*)

pprint(*object*)

robot.utils.utf8reader module

class robot.utils.utf8reader.**Utf8Reader**(*path_or_file*)
Bases: object

read()

readlines()

robot.variables package

Implements storing and resolving variables.

This package is mainly for internal usage.

Submodules

robot.variables.assigner module

class robot.variables.assigner.**VariableAssignment**(*assignment*)
Bases: object

validate_assignment()

assigner(*context*)

class robot.variables.assigner.**AssignmentValidator**
Bases: object

validate(*variable*)

class robot.variables.assigner.**VariableAssigner**(*assignment*, *context*)
Bases: object

assign(*return_value*)

robot.variables.assigner.**ReturnValueResolver**(*assignment*)

class robot.variables.assigner.**NoReturnValueResolver**
Bases: object

resolve(*return_value*)

class robot.variables.assigner.**OneReturnValueResolver**(*variable*)
Bases: object

resolve(*return_value*)

class robot.variables.assigner.**ScalarsOnlyReturnValueResolver**(*variables*)
Bases: robot.variables.assigner._MultiReturnValueResolver

resolve(*return_value*)

class robot.variables.assigner.**ScalarsAndListReturnValueResolver**(*variables*)
Bases: robot.variables.assigner._MultiReturnValueResolver

resolve(*return_value*)

robot.variables.filesetter module

class robot.variables.filesetter.**VariableFileSetter**(*store*)
Bases: object

set(*path_or_variables*, *args=None*, *overwrite=False*)

class robot.variables.filesetter.**YamlImporter**
Bases: object

import_variables(*path*, *args=None*)

class robot.variables.filesetter.**PythonImporter**
Bases: object

import_variables(*path*, *args=None*)

robot.variables.finders module

robot.variables.finders.**get_java_property**(*name*)

robot.variables.finders.**get_java_properties**()

class robot.variables.finders.**VariableFinder**(*variable_store*)
Bases: object

find(*name*)

class robot.variables.finders.**StoredFinder**(*store*)
Bases: object

identifiers = '$@&'

find(*name*)

class robot.variables.finders.**NumberFinder**
Bases: object

identifiers = '$'

find(*name*)

class robot.variables.finders.**EmptyFinder**
Bases: object

identifiers = '$@&'

classmethod find(*key*)

class robot.variables.finders.**ExtendedFinder**(*finder*)
Bases: object

identifiers = '$@&'

find(*name*)

class robot.variables.finders.**EnvironmentFinder**
Bases: object

identifiers = '%'

find(*name*)

robot.variables.isvar module

robot.variables.isvar.**is_var**(*string*, *identifiers='$@&'*)

robot.variables.isvar.**is_scalar_var**(*string*)

robot.variables.isvar.**is_list_var**(*string*)

robot.variables.isvar.**is_dict_var**(*string*)

robot.variables.isvar.**contains_var**(*string*, *identifiers='$@&'*)

robot.variables.isvar.**validate_var**(*string*, *identifiers='$@&'*)

robot.variables.notfound module

robot.variables.notfound.**variable_not_found**(*name*, *candidates*, *msg=None*, *deco_braces=True*)
Raise DataError for missing variable name.

Return recommendations for similar variable names if any are found.

robot.variables.replacer module

class robot.variables.replacer.**VariableReplacer**(*variables*)
Bases: object

replace_list(*items*, *replace_until=None*, *ignore_errors=False*)
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_until' can be used to limit replacing arguments to certain index from the beginning. Used with Run Keyword variants that only want to resolve some of the arguments in the beginning and pass others to called keywords unmodified.

replace_scalar(*item*, *ignore_errors=False*)
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*, *ignore_errors=False*)
Replaces variables from a string. Result is always a string.

robot.variables.scopes module

class robot.variables.scopes.**VariableScopes**(*settings*)
Bases: object

current

start_suite()

end_suite()

start_test()

end_test()

start_keyword()

end_keyword()

replace_list(*items*, *replace_until=None*, *ignore_errors=False*)

replace_scalar(*items*, *ignore_errors=False*)

replace_string(*string*, *ignore_errors=False*)

set_from_file(*path*, *args*, *overwrite=False*)

set_from_variable_table(*variables*, *overwrite=False*)

resolve_delayed()

set_global(*name*, *value*)

set_suite(*name*, *value*, *top=False*, *children=False*)

set_test(*name*, *value*)

set_keyword(*name*, *value*)

as_dict(*decoration=True*)

class robot.variables.scopes.**GlobalVariables**(*settings*)
Bases: *robot.variables.variables.Variables*

as_dict(*decoration=True*)

clear()

copy()

replace_list(*items*, *replace_until=None*, *ignore_errors=False*)

replace_scalar(*item*, *ignore_errors=False*)

replace_string(*item*, *ignore_errors=False*)

resolve_delayed()

set_from_file(*path_or_variables*, *args=None*, *overwrite=False*)

set_from_variable_table(*variables*, *overwrite=False*)

update(*variables*)

class robot.variables.scopes.**SetVariables**
Bases: object

start_suite()

end_suite()

start_test()

end_test()

start_keyword()

end_keyword()

set_global(*name*, *value*)

set_suite(*name*, *value*)

set_test(*name*, *value*)

set_keyword(*name*, *value*)

update(*variables*)

robot.variables.splitter module

class robot.variables.splitter.**VariableSplitter**(*string*, *identifiers='$@%&*'*)
Bases: object

get_replaced_variable(*replacer*)

is_variable()

is_list_variable()

is_dict_variable()

class robot.variables.splitter.**VariableIterator**(*string*, *identifiers='$@%&*'*)
Bases: object

robot.variables.store module

class robot.variables.store.**VariableStore**(*variables*)
Bases: object

resolve_delayed()

update(*store*)

clear()

add(*name*, *value*, *overwrite=True*, *decorated=True*)

remove(*name*)

as_dict(*decoration=True*)

robot.variables.tablesetter module

class robot.variables.tablesetter.**VariableTableSetter**(*store*)
Bases: object

set(*variables*, *overwrite=False*)

class robot.variables.tablesetter.**VariableTableReader**
Bases: object

read(*variables*)

robot.variables.tablesetter.**VariableTableValue**(*value*, *name*, *error_reporter=None*)

class robot.variables.tablesetter.**VariableTableValueBase**(*values*, *error_reporter=None*)
Bases: object

resolve(*variables*)

report_error(*error*)

class robot.variables.tablesetter.**ScalarVariableTableValue**(*values*, *error_reporter=None*)
Bases: *robot.variables.tablesetter.VariableTableValueBase*

report_error(*error*)

resolve(*variables*)

class robot.variables.tablesetter.**ListVariableTableValue**(*values*, *error_reporter=None*)
Bases: *robot.variables.tablesetter.VariableTableValueBase*

report_error(*error*)

resolve(*variables*)

class robot.variables.tablesetter.**DictVariableTableValue**(*values*, *error_reporter=None*)
Bases: *robot.variables.tablesetter.VariableTableValueBase*

report_error(*error*)

resolve(*variables*)

robot.variables.variables module

class robot.variables.variables.**Variables**
Bases: object

Represents a set of variables.

Contains methods for replacing variables from list, scalars, and strings. On top of ${scalar}, @{list} and &{dict} variables, these methods handle also %{environment} variables.

resolve_delayed()

replace_list(*items*, *replace_until=None*, *ignore_errors=False*)

replace_scalar(*item*, *ignore_errors=False*)

replace_string(*item*, *ignore_errors=False*)

set_from_file(*path_or_variables*, *args=None*, *overwrite=False*)

set_from_variable_table(*variables*, *overwrite=False*)

clear()

copy()

update(*variables*)

as_dict(*decoration=True*)

robot.writer package

Implements writing of parsed, and possibly edited, test data back to files.

This functionality is used by *robot.parsing.model.TestCaseFile.save()* and indirectly by *robot.tidy*. External tools should not need to use this package directly.

This package is considered stable, although the planned changes to *robot.parsing* may affect also this package.

Submodules

robot.writer.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*)

robot.writer.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*)

robot.writer.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')
```

txt_format = 'txt'

html_format = 'html'

tsv_format = 'tsv'

robot_format = 'robot'

txt_column_count = 8

html_column_count = 5

tsv_column_count = 8

robot.writer.filewriters module

robot.writer.filewriters.**FileWriter**(*context*)
Creates and returns a FileWriter object.

Parameters **context** (*WritingContext*) – The type of the 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*)

robot.writer.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*)

robot.writer.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*

robot.writer.htmltemplate module

robot.writer.rowsplitter module

class robot.writer.rowsplitter.**RowSplitter**(*cols=8*, *split_multiline_doc=True*)
Bases: object

split(*row*, *table_type*)

4.1.2 Submodules

4.1.3 robot.errors module

Exceptions and return codes used internally.

External libraries should not used exceptions defined here.

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 caught by keywords that run other keywords (e.g. *Run Keyword And Expect Error*).

args

message

exception robot.errors.**VariableError**(*message=''*, *details=''*)
Bases: *robot.errors.DataError*

Used when variable does not exist.

VariableErrors are caught by keywords that run other keywords (e.g. *Run Keyword And Expect Error*).

args

message

exception robot.errors.**KeywordError**(*message=''*, *details=''*)
Bases: *robot.errors.DataError*

Used when no keyword is found or there is more than one match.

KeywordErrors are caught by keywords that run other keywords (e.g. *Run Keyword And Expect Error*).

args

message

exception robot.errors.**TimeoutError**(*message=''*, *test_timeout=True*)
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*).

keyword_timeout

args

message

exception robot.errors.**Information**(*message=''*, *details=''*)
Bases: *robot.errors.RobotError*

Used by argument parser with –help or –version.

args

message

exception robot.errors.**ExecutionStatus**(*message*, *test_timeout=False*, *keyword_timeout=False*, *syntax=False*, *exit=False*, *continue_on_failure=False*, *return_value=None*)
Bases: *robot.errors.RobotError*

Base class for exceptions communicating status in test execution.

timeout

dont_continue

continue_on_failure

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

get_errors()

status

args

message

exception robot.errors.**ExecutionFailed**(*message*, *test_timeout=False*, *keyword_timeout=False*, *syntax=False*, *exit=False*, *continue_on_failure=False*, *return_value=None*)

Bases: *robot.errors.ExecutionStatus*

Used for communicating failures in test execution.

args

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

continue_on_failure

dont_continue

get_errors()

message

status

timeout

exception robot.errors.**HandlerExecutionFailed**(*details*)

Bases: *robot.errors.ExecutionFailed*

args

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

continue_on_failure

dont_continue

get_errors()

message

status

timeout

exception robot.errors.**ExecutionFailures**(*errors*, *message=None*)

Bases: *robot.errors.ExecutionFailed*

get_errors()

args

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

continue_on_failure

dont_continue

message

status

timeout

exception robot.errors.**UserKeywordExecutionFailed**(*run_errors=None*, *teardown_errors=None*)

Bases: *robot.errors.ExecutionFailures*

args

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

continue_on_failure

dont_continue

get_errors()

message

status

timeout

exception robot.errors.**ExecutionPassed**(*message=None*, ***kwargs*)

Bases: *robot.errors.ExecutionStatus*

Base class for all exceptions communicating that execution passed.

Should not be raised directly, but more detailed exceptions used instead.

set_earlier_failures(*failures*)

earlier_failures

status

args

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

continue_on_failure

dont_continue

get_errors()

message

timeout

exception robot.errors.**PassExecution**(*message*)

Bases: *robot.errors.ExecutionPassed*

Used by 'Pass Execution' keyword.

args

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

continue_on_failure

dont_continue

earlier_failures

get_errors()

message

set_earlier_failures(*failures*)

status

timeout

exception robot.errors.**ContinueForLoop**(*message=None*, ***kwargs*)
Bases: *robot.errors.ExecutionPassed*

Used by 'Continue For Loop' keyword.

args

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

continue_on_failure

dont_continue

earlier_failures

get_errors()

message

set_earlier_failures(*failures*)

status

timeout

exception robot.errors.**ExitForLoop**(*message=None*, ***kwargs*)
Bases: *robot.errors.ExecutionPassed*

Used by 'Exit For Loop' keyword.

args

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

continue_on_failure

dont_continue

earlier_failures

get_errors()

message

set_earlier_failures(*failures*)

status

timeout

exception robot.errors.**ReturnFromKeyword**(*return_value=None*, *failures=None*)
Bases: *robot.errors.ExecutionPassed*

Used by 'Return From Keyword' keyword.

args

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

continue_on_failure

dont_continue

earlier_failures

get_errors()

message

set_earlier_failures(*failures*)

status

timeout

exception robot.errors.**RemoteError**(*message=''*, *details=''*, *fatal=False*, *continuable=False*)
Bases: *robot.errors.RobotError*

Used by Remote library to report remote errors.

args

message

4.1.4 robot.jarrunner module

4.1.5 robot.libdoc module

Module implementing the command line entry point for the Libdoc tool.

This module can be executed from the command line using the following approaches:

```
python -m robot.libdoc
python path/to/robot/libdoc.py
```

Instead of `python` it is possible to use also other Python interpreters.

This module also provides *libdoc()* and *libdoc_cli()* functions that can be used programmatically. Other code is for internal usage.

Libdoc itself is implemented in the *libdocpkg* package.

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*, *exit=True*)

parse_arguments(*cli_args*)
Public interface for parsing command line arguments.

Parameters **cli_args** – Command line arguments as a list

Returns options (dict), arguments (list)

Raises *Information* when –help or –version used

Raises *DataError* when parsing fails

robot.libdoc.**libdoc_cli**(*arguments*)
Executes Libdoc similarly as from the command line.

Parameters **arguments** – Command line arguments as a list of strings.

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

Example:

```
from robot.libdoc import libdoc_cli

libdoc_cli(['--version', '1.0', 'MyLibrary.py', 'MyLibraryDoc.html'])
```

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

Executes Libdoc.

Parameters

- **library_or_resource** – Name or path of the test library or resource file to be documented.
- **outfile** – Path path to the file where to write outputs.
- **name** – Custom name to give to the documented library or resource.
- **version** – Version to give to the documented library or resource.
- **format** – Specifies whether to generate HTML or XML output. If this options is not used, the format is got from the extension of the output file. Possible values are `'HTML'` and `'XML'`.
- **docformat** – Documentation source format. Possible values are `'ROBOT'`, `'reST'`, `'HTML'` and `'TEXT'`. The default value can be specified in test library source code and the initial default is `'ROBOT'`. New in Robot Framework 3.0.3.

Arguments have same semantics as Libdoc command line options with same names. Run `python -m robot.libdoc --help` or consult the Libdoc section in the Robot Framework User Guide for more details.

Example:

```
from robot.libdoc import libdoc

libdoc('MyLibrary.py', 'MyLibraryDoc.html', version='1.0')
```

4.1.6 robot.pythonpathsetter module

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

robot.pythonpathsetter.**add_path**(*path*, *end=False*)

robot.pythonpathsetter.**remove_path**(*path*)

4.1.7 robot.rebot module

Module implementing the command line entry point for post-processing outputs.

This module can be executed from the command line using the following approaches:

```
python -m robot.rebot
python path/to/robot/rebot.py
```

Instead of `python` it is possible to use also other Python interpreters. This module is also used by the installed `rebot` start-up script.

This module also provides *rebot()* and *rebot_cli()* functions that can be used programmatically. Other code is for internal usage.

class robot.rebot.**Rebot**

Bases: *robot.run.RobotFramework*

main(*datasources*, ***options*)

console(*msg*)

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

execute_cli(*cli_arguments*, *exit=True*)

parse_arguments(*cli_args*)

Public interface for parsing command line arguments.

Parameters **cli_args** – Command line arguments as a list

Returns options (dict), arguments (list)

Raises *Information* when –help or –version used

Raises *DataError* when parsing fails

validate(*options*, *arguments*)

robot.rebot.**rebot_cli**(*arguments=None*, *exit=True*)

Command line execution entry point for post-processing outputs.

Parameters

- **arguments** – Command line options and arguments as a list of strings. Starting from RF 3.1, defaults to `sys.argv[1:]` if not given.
- **exit** – If `True`, call `sys.exit` with the return code denoting execution status, otherwise just return the rc. New in RF 3.0.1.

Entry point used when post-processing outputs from the command line, but can also be used by custom scripts. Especially useful if the script itself needs to accept same arguments as accepted by Rebot, because the script can just pass them forward directly along with the possible default values it sets itself.

Example:

```
from robot import rebot_cli

rebot_cli(['--name', 'Example', '--log', 'NONE', 'o1.xml', 'o2.xml'])
```

See also the *rebot()* function that allows setting options as keyword arguments like `name="Example"` and generally has a richer API for programmatic Rebot execution.

robot.rebot.**rebot**(**outputs*, ***options*)

Programmatic entry point for post-processing outputs.

Parameters

- **outputs** – Paths to Robot Framework output files similarly as when running the `rebot` command on the command line.
- **options** – Options to configure processing outputs. Accepted options are mostly same as normal command line options to the `rebot` command. Option names match command line option long names without hyphens so that, for example, `--name` becomes `name`.

The semantics related to passing options are exactly the same as with the *run()* function. See its documentation for more details.

Examples:

```
from robot import rebot

rebot('path/to/output.xml')
with open('stdout.txt', 'w') as stdout:
    rebot('o1.xml', 'o2.xml', name='Example', log=None, stdout=stdout)
```

Equivalent command line usage:

```
rebot path/to/output.xml
rebot --name Example --log NONE o1.xml o2.xml > stdout.txt
```

4.1.8 robot.run module

Module implementing the command line entry point for executing tests.

This module can be executed from the command line using the following approaches:

```
python -m robot.run
python path/to/robot/run.py
```

Instead of `python` it is possible to use also other Python interpreters. This module is also used by the installed `robot` start-up script.

This module also provides *run()* and *run_cli()* functions that can be used programmatically. Other code is for internal usage.

class robot.run.**RobotFramework**

Bases: *robot.utils.application.Application*

main(*datasources*, ***options*)

validate(*options*, *arguments*)

console(*msg*)

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

execute_cli(*cli_arguments*, *exit=True*)

parse_arguments(*cli_args*)

Public interface for parsing command line arguments.

Parameters **cli_args** – Command line arguments as a list

Returns options (dict), arguments (list)

Raises *Information* when –help or –version used

Raises *DataError* when parsing fails

robot.run.**run_cli**(*arguments=None*, *exit=True*)

Command line execution entry point for running tests.

Parameters

- **arguments** – Command line options and arguments as a list of strings. Starting from RF 3.1, defaults to `sys.argv[1:]` if not given.
- **exit** – If `True`, call `sys.exit` with the return code denoting execution status, otherwise just return the rc. New in RF 3.0.1.

Entry point used when running tests from the command line, but can also be used by custom scripts that execute tests. Especially useful if the script itself needs to accept same arguments as accepted by Robot Framework, because the script can just pass them forward directly along with the possible default values it sets itself.

Example:

```
from robot import run_cli

# Run tests and return the return code.
rc = run_cli(['--name', 'Example', 'tests.robot'], exit=False)

# Run tests and exit to the system automatically.
run_cli(['--name', 'Example', 'tests.robot'])
```

See also the *run()* function that allows setting options as keyword arguments like `name="Example"` and generally has a richer API for programmatic test execution.

`robot.run.run`(**tests*, ***options*)

Programmatic entry point for running tests.

Parameters

- **tests** – Paths to test case files/directories to be executed similarly as when running the `robot` command on the command line.
- **options** – Options to configure and control execution. Accepted options are mostly same as normal command line options to the `robot` command. Option names match command line option long names without hyphens so that, for example, `--name` becomes `name`.

Most options that can be given from the command line work. An exception is that options `--pythonpath`, `--argumentfile`, `--help` and `--version` are not supported.

Options that can be given on the command line multiple times can be passed as lists. For example, `include=['tag1', 'tag2']` is equivalent to `--include tag1 --include tag2`. If such options are used only once, they can be given also as a single string like `include='tag'`.

Options that accept no value can be given as Booleans. For example, `dryrun=True` is same as using the `--dryrun` option.

Options that accept string `NONE` as a special value can also be used with Python `None`. For example, using `log=None` is equivalent to `--log NONE`.

`listener`, `prerunmodifier` and `prerebotmodifier` options allow passing values as Python objects in addition to module names these command line options support. For example, `run('tests', listener=MyListener())`.

To capture the standard output and error streams, pass an open file or file-like object as special keyword arguments `stdout` and `stderr`, respectively.

A return code is returned similarly as when running on the command line. Zero means that tests were executed and no critical test failed, values up to 250 denote the number of failed critical tests, and values between 251-255 are for other statuses documented in the Robot Framework User Guide.

Example:

```
from robot import run

run('path/to/tests.robot')
run('tests.robot', include=['tag1', 'tag2'], splitlog=True)
with open('stdout.txt', 'w') as stdout:
    run('t1.robot', 't2.robot', name='Example', log=None, stdout=stdout)
```

Equivalent command line usage:

```
robot path/to/tests.robot
robot --include tag1 --include tag2 --splitlog tests.robot
robot --name Example --log NONE t1.robot t2.robot > stdout.txt
```

4.1.9 robot.testdoc module

Module implementing the command line entry point for the *Testdoc* tool.

This module can be executed from the command line using the following approaches:

```
python -m robot.testdoc
python path/to/robot/testdoc.py
```

Instead of `python` it is possible to use also other Python interpreters.

This module also provides *testdoc()* and *testdoc_cli()* functions that can be used programmatically. Other code is for internal usage.

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

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

console(*msg*)

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

execute_cli(*cli_arguments*, *exit=True*)

parse_arguments(*cli_args*)
Public interface for parsing command line arguments.

Parameters **cli_args** – Command line arguments as a list

Returns options (dict), arguments (list)

Raises *Information* when –help or –version used

Raises *DataError* when parsing fails

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**(*arguments*)
Executes *Testdoc* similarly as from the command line.

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

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

Example:

```
from robot.testdoc import testdoc_cli

testdoc_cli(['--title', 'Test Plan', 'mytests', 'plan.html'])
```

`robot.testdoc.`**`testdoc`**(**arguments*, ***options*)

Executes *Testdoc* programmatically.

Arguments and options have same semantics, and options have same names, as arguments and options to Testdoc.

Example:

```
from robot.testdoc import testdoc

testdoc('mytests', 'plan.html', title='Test Plan')
```

4.1.10 robot.tidy module

Module implementing the command line entry point for the *Tidy* tool.

This module can be executed from the command line using the following approaches:

```
python -m robot.tidy
python path/to/robot/tidy.py
```

Instead of `python` it is possible to use also other Python interpreters.

This module also provides *Tidy* class and *tidy_cli()* function that can be used programmatically. Other code is for internal usage.

`class` `robot.tidy.`**`Tidy`**(*format='txt'*, *use_pipes=False*, *space_count=4*, *line_separator='n'*)

Bases: `object`

Programmatic API for the *Tidy* tool.

Arguments accepted when creating an instance have same semantics as Tidy command line options with same names.

`file`(*path*, *output=None*)

Tidy a file.

Parameters

- **`path`** – Path of the input file.
- **`output`** – Path of the output file. If not given, output is returned.

Use *inplace()* to tidy files in-place.

`inplace`(**paths*)

Tidy file(s) in-place.

Parameters **`paths`** – Paths of the files to to process.

`directory`(*path*)

Tidy a directory.

Parameters **path** – Path of the directory to process.

All files in a directory, recursively, are processed in-place.

class robot.tidy.**TidyCommandLine**
Bases: *robot.utils.application.Application*

Command line interface for the *Tidy* tool.

Typically *tidy_cli()* is a better suited for command line style usage and *Tidy* for other programmatic usage.

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*, *exit=True*)

parse_arguments(*cli_args*)
Public interface for parsing command line arguments.

Parameters **cli_args** – Command line arguments as a list

Returns options (dict), arguments (list)

Raises *Information* when –help or –version used

Raises *DataError* when parsing fails

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**(*arguments*)
Executes *Tidy* similarly as from the command line.

Parameters **arguments** – Command line arguments as a list of strings.

Example:

```
from robot.tidy import tidy_cli

tidy_cli(['--format', 'txt', 'tests.html'])
```

4.1.11 robot.version module

robot.version.**get_version**(*naked=False*)

robot.version.**get_full_version**(*program=None*, *naked=False*)

robot.version.**get_interpreter**()

CHAPTER 5

Indices

- genindex
- modindex
- search

Python Module Index

r

robot, 9
robot.api, 7
robot.api.deco, 12
robot.api.logger, 13
robot.conf, 15
robot.conf.gatherfailed, 15
robot.conf.settings, 17
robot.errors, 306
robot.htmldata, 19
robot.htmldata.htmlfilewriter, 19
robot.htmldata.jsonwriter, 20
robot.htmldata.normaltemplate, 20
robot.htmldata.template, 21
robot.libdoc, 311
robot.libdocpkg, 21
robot.libdocpkg.builder, 21
robot.libdocpkg.consoleviewer, 21
robot.libdocpkg.htmlwriter, 21
robot.libdocpkg.javabuilder, 22
robot.libdocpkg.model, 22
robot.libdocpkg.output, 22
robot.libdocpkg.robotbuilder, 22
robot.libdocpkg.specbuilder, 23
robot.libdocpkg.writer, 23
robot.libdocpkg.xmlwriter, 23
robot.libraries, 23
robot.libraries.BuiltIn, 23
robot.libraries.Collections, 47
robot.libraries.DateTime, 53
robot.libraries.Dialogs, 58
robot.libraries.dialogs_py, 104
robot.libraries.Easter, 59
robot.libraries.OperatingSystem, 59
robot.libraries.Process, 69
robot.libraries.Remote, 75
robot.libraries.Reserved, 77
robot.libraries.Screenshot, 77
robot.libraries.String, 78
robot.libraries.Telnet, 84
robot.libraries.XML, 93
robot.model, 169
robot.model.configurer, 170
robot.model.criticality, 171
robot.model.filter, 171
robot.model.imports, 173
robot.model.itemlist, 174
robot.model.keyword, 174
robot.model.message, 176
robot.model.metadata, 177
robot.model.modelobject, 178
robot.model.modifier, 178
robot.model.namepatterns, 179
robot.model.statistics, 179
robot.model.stats, 181
robot.model.suitestatistics, 183
robot.model.tags, 183
robot.model.tagsetter, 184
robot.model.tagstatistics, 185
robot.model.testcase, 186
robot.model.testsuite, 187
robot.model.totalstatistics, 189
robot.model.visitor, 190
robot.output, 192
robot.output.console, 192
robot.output.console.dotted, 192
robot.output.console.highlighting, 194
robot.output.console.quiet, 194
robot.output.console.verbose, 195
robot.output.debugfile, 195
robot.output.filelogger, 195
robot.output.librarylogger, 196
robot.output.listenerarguments, 196
robot.output.listenermethods, 197
robot.output.listeners, 198
robot.output.logger, 198
robot.output.loggerhelper, 199
robot.output.output, 201
robot.output.pyloggingconf, 201

robot.output.stdoutlogsplitter, 203
robot.output.xmllogger, 203
robot.parsing, 204
robot.parsing.comments, 205
robot.parsing.datarow, 205
robot.parsing.htmlreader, 206
robot.parsing.model, 207
robot.parsing.populators, 213
robot.parsing.restreader, 213
robot.parsing.restsupport, 213
robot.parsing.settings, 214
robot.parsing.tablepopulators, 218
robot.parsing.tsvreader, 220
robot.parsing.txtreader, 220
robot.pythonpathsetter, 312
robot.rebot, 312
robot.reporting, 220
robot.reporting.jsbuildingcontext, 220
robot.reporting.jsexecutionresult, 221
robot.reporting.jsmodelbuilders, 221
robot.reporting.jswriter, 222
robot.reporting.logreportwriters, 222
robot.reporting.outputwriter, 222
robot.reporting.resultwriter, 224
robot.reporting.stringcache, 224
robot.reporting.xunitwriter, 225
robot.result, 227
robot.result.configurer, 228
robot.result.executionerrors, 229
robot.result.executionresult, 229
robot.result.flattenkeywordmatcher, 232
robot.result.keywordremover, 232
robot.result.merger, 238
robot.result.messagefilter, 239
robot.result.model, 240
robot.result.resultbuilder, 247
robot.result.suiteteardownfailed, 249
robot.result.visitor, 251
robot.result.xmlelementhandlers, 253
robot.run, 314
robot.running, 256
robot.running.arguments, 258
robot.running.arguments.argumentmapper, 258
robot.running.arguments.argumentparser, 258
robot.running.arguments.argumentresolver, 259
robot.running.arguments.argumentspec, 259
robot.running.arguments.argumentvalidator, 260
robot.running.arguments.embedded, 260
robot.running.builder, 261
robot.running.context, 262
robot.running.defaults, 262
robot.running.dynamicmethods, 262
robot.running.handlers, 263
robot.running.handlerstore, 263
robot.running.importer, 263
robot.running.librarykeywordrunner, 264
robot.running.libraryscopes, 264
robot.running.model, 265
robot.running.namespace, 272
robot.running.outputcapture, 273
robot.running.randomizer, 273
robot.running.runkwregister, 274
robot.running.runner, 274
robot.running.signalhandler, 275
robot.running.status, 276
robot.running.statusreporter, 277
robot.running.steprunner, 277
robot.running.testlibraries, 278
robot.running.timeouts, 260
robot.running.timeouts.posix, 261
robot.running.timeouts.windows, 261
robot.running.usererrorhandler, 278
robot.running.userkeyword, 278
robot.running.userkeywordrunner, 279
robot.testdoc, 316
robot.tidy, 317
robot.utils, 279
robot.utils.application, 280
robot.utils.argumentparser, 280
robot.utils.asserts, 281
robot.utils.charwidth, 283
robot.utils.compat, 284
robot.utils.compress, 284
robot.utils.connectioncache, 284
robot.utils.dotdict, 285
robot.utils.encoding, 286
robot.utils.encodingsniffer, 286
robot.utils.error, 286
robot.utils.escaping, 287
robot.utils.etreewrapper, 287
robot.utils.frange, 287
robot.utils.htmlformatters, 287
robot.utils.importer, 289
robot.utils.markuputils, 290
robot.utils.markupwriters, 290
robot.utils.match, 291
robot.utils.misc, 291
robot.utils.normalizing, 292
robot.utils.platform, 293
robot.utils.recommendations, 293
robot.utils.robotenv, 293
robot.utils.robotinspect, 293
robot.utils.robotio, 293

robot.utils.robotpath, 293
robot.utils.robottime, 294
robot.utils.robottypes, 295
robot.utils.robottypes2, 295
robot.utils.setter, 296
robot.utils.sortable, 296
robot.utils.text, 296
robot.utils.unic, 296
robot.utils.utf8reader, 297
robot.variables, 297
robot.variables.assigner, 297
robot.variables.filesetter, 298
robot.variables.finders, 298
robot.variables.isvar, 299
robot.variables.notfound, 299
robot.variables.replacer, 299
robot.variables.scopes, 300
robot.variables.splitter, 301
robot.variables.store, 301
robot.variables.tablesetter, 302
robot.variables.variables, 302
robot.version, 318
robot.writer, 303
robot.writer.aligners, 303
robot.writer.dataextractor, 303
robot.writer.datafilewriter, 303
robot.writer.filewriters, 305
robot.writer.formatters, 305
robot.writer.htmlformatter, 305
robot.writer.htmltemplate, 306
robot.writer.rowsplitter, 306

Index

A

abspath() (*in module robot.utils.robotpath*), 293
AbstractLogger (*class in robot.output.loggerhelper*), 199
AbstractLoggerProxy (*class in robot.output.loggerhelper*), 200
accept_gzip_encoding (*robot.libraries.Remote.TimeoutHTTPSTransport attribute*), 76
accept_gzip_encoding (*robot.libraries.Remote.TimeoutHTTPTransport attribute*), 76
acquire() (*robot.output.pyloggingconf.RobotHandler method*), 201
active (*robot.running.timeouts.KeywordTimeout attribute*), 260
active (*robot.running.timeouts.TestTimeout attribute*), 260
add() (*robot.model.tags.Tags method*), 183
add() (*robot.parsing.comments.CommentCache method*), 205
add() (*robot.parsing.comments.Comments method*), 205
add() (*robot.parsing.model.KeywordTable method*), 211
add() (*robot.parsing.model.TestCaseTable method*), 210
add() (*robot.parsing.model.VariableTable method*), 210
add() (*robot.parsing.populators.FromFilePopulator method*), 213
add() (*robot.parsing.settings.ImportList method*), 218
add() (*robot.parsing.settings.MetadataList method*), 218
add() (*robot.parsing.tablepopulators.DocumentationPopulator method*), 219
add() (*robot.parsing.tablepopulators.ForLoopPopulator method*), 219
add() (*robot.parsing.tablepopulators.KeywordTablePopulator method*), 219
add() (*robot.parsing.tablepopulators.MetadataPopulator method*), 219
add() (*robot.parsing.tablepopulators.NullPopulator method*), 218
add() (*robot.parsing.tablepopulators.Populator method*), 218
add() (*robot.parsing.tablepopulators.SettingPopulator method*), 219
add() (*robot.parsing.tablepopulators.SettingTablePopulator method*), 218
add() (*robot.parsing.tablepopulators.StepPopulator method*), 219
add() (*robot.parsing.tablepopulators.TestCasePopulator method*), 219
add() (*robot.parsing.tablepopulators.TestTablePopulator method*), 218
add() (*robot.parsing.tablepopulators.UserKeywordPopulator method*), 219
add() (*robot.parsing.tablepopulators.VariablePopulator method*), 219
add() (*robot.parsing.tablepopulators.VariableTablePopulator method*), 218
add() (*robot.reporting.stringcache.StringCache method*), 225
add() (*robot.result.executionerrors.ExecutionErrors method*), 229
add() (*robot.running.handlerstore.HandlerStore method*), 263
add() (*robot.running.importer.ImportCache method*), 264
add() (*robot.utils.htmlformatters.HeaderFormatter method*), 288
add() (*robot.utils.htmlformatters.ListFormatter method*), 289
add() (*robot.utils.htmlformatters.ParagraphFormatter method*), 288
add() (*robot.utils.htmlformatters.PreformattedFormatter method*), 289
add() (*robot.utils.htmlformatters.RulerFormatter method*), 288

add() (*robot.utils.htmlformatters.TableFormatter method*), 288
add() (*robot.variables.store.VariableStore method*), 301
add_child() (*robot.parsing.model.TestDataDirectory method*), 209
add_data() (*robot.parsing.restsupport.RobotDataStorage method*), 214
add_element() (*robot.libraries.XML.XML method*), 102
add_for_loop() (*robot.parsing.model.TestCase method*), 211
add_for_loop() (*robot.parsing.model.UserKeyword method*), 211
add_library() (*robot.parsing.model.InitFileSettingTable method*), 210
add_library() (*robot.parsing.model.ResourceFileSettingTable method*), 209
add_library() (*robot.parsing.model.TestCaseFileSettingTable method*), 209
add_metadata() (*robot.parsing.model.InitFileSettingTable method*), 210
add_metadata() (*robot.parsing.model.ResourceFileSettingTable method*), 209
add_metadata() (*robot.parsing.model.TestCaseFileSettingTable method*), 209
add_name() (*robot.parsing.restsupport.CaptureRobotData method*), 213
add_path() (*in module robot.pythonpathsetter*), 312
add_resource() (*robot.parsing.model.InitFileSettingTable method*), 210
add_resource() (*robot.parsing.model.ResourceFileSettingTable method*), 209
add_resource() (*robot.parsing.model.TestCaseFileSettingTable method*), 209
add_result() (*robot.result.executionresult.CombinedResult method*), 230
add_stat() (*robot.model.stats.SuiteStat method*), 181
add_step() (*robot.parsing.model.ForLoop method*), 212
add_step() (*robot.parsing.model.TestCase method*), 211
add_step() (*robot.parsing.model.UserKeyword method*), 211
add_tags (*robot.model.configurer.SuiteConfigurer attribute*), 170
add_tags (*robot.result.configurer.SuiteConfigurer attribute*), 228
add_test() (*robot.model.stats.CombinedTagStat method*), 182
add_test() (*robot.model.stats.CriticalTagStat method*), 182
add_test() (*robot.model.stats.Stat method*), 181
add_test() (*robot.model.stats.SuiteStat method*), 181
add_test() (*robot.model.stats.TagStat method*), 182
add_test() (*robot.model.stats.TotalStat method*), 181
add_test() (*robot.model.suitestatistics.SuiteStatisticsBuilder method*), 183
add_test() (*robot.model.tagstatistics.TagStatisticsBuilder method*), 185
add_test() (*robot.model.totalstatistics.TotalStatisticsBuilder method*), 189
add_time_to_date() (*in module robot.libraries.DateTime*), 57
add_time_to_time() (*in module robot.libraries.DateTime*), 58
add_variables() (*robot.parsing.model.InitFileSettingTable method*), 210
add_variables() (*robot.parsing.model.ResourceFileSettingTable method*), 209
add_variables() (*robot.parsing.model.TestCaseFileSettingTable method*), 209
addFilter() (*robot.output.pyloggingconf.RobotHandler method*), 201
after() (*robot.libraries.dialogs_py.InputDialog method*), 117
after() (*robot.libraries.dialogs_py.MessageDialog method*), 104
after() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 143
after() (*robot.libraries.dialogs_py.PassFailDialog method*), 156
after() (*robot.libraries.dialogs_py.SelectionDialog method*), 130
after_cancel() (*robot.libraries.dialogs_py.InputDialog method*), 118
after_cancel() (*robot.libraries.dialogs_py.MessageDialog method*), 104
after_cancel() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 144
after_cancel() (*robot.libraries.dialogs_py.PassFailDialog method*), 157
after_cancel() (*robot.libraries.dialogs_py.SelectionDialog method*), 131
after_idle() (*robot.libraries.dialogs_py.InputDialog method*), 118
after_idle() (*robot.libraries.dialogs_py.MessageDialog method*), 104
after_idle() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 144
after_idle() (*robot.libraries.dialogs_py.PassFailDialog method*), 157
after_idle() (*robot.libraries.dialogs_py.SelectionDialog method*), 131
align_row() (*robot.writer.aligners.ColumnAligner method*), 303
align_row() (*robot.writer.aligners.FirstColumnAligner method*), 303
align_row() (*robot.writer.aligners.NullAligner*

method), 303
align_rows() (*robot.writer.aligners.ColumnAligner method*), 303
align_rows() (*robot.writer.aligners.FirstColumnAligner method*), 303
align_rows() (*robot.writer.aligners.NullAligner method*), 303
all (*robot.model.keyword.Keywords attribute*), 176
all (*robot.model.totalstatistics.TotalStatistics attribute*), 189
all (*robot.parsing.datarow.DataRow attribute*), 205
all_tags (*robot.libdocpkg.model.LibraryDoc attribute*), 22
AllKeywordsRemover (*class in robot.result.keywordremover*), 232
ALLOWED_TYPES (*robot.model.imports.Import attribute*), 173
also_teardown_message (*robot.running.status.ParentMessage attribute*), 277
also_teardown_message (*robot.running.status.SuiteMessage attribute*), 277
also_teardown_message (*robot.running.status.TestMessage attribute*), 276
AnchorNameCell (*class in robot.writer.htmlformatter*), 306
AndTagPattern (*class in robot.model.tags*), 183
AnsiHighlighter (*class in robot.output.console.highlighting*), 194
any_timeout_occurred() (*robot.running.timeouts.TestTimeout method*), 260
append() (*robot.model.imports.Imports method*), 173
append() (*robot.model.itemlist.ItemList method*), 174
append() (*robot.model.keyword.Keywords method*), 175
append() (*robot.model.message.Messages method*), 177
append() (*robot.model.testcase.TestCases method*), 187
append() (*robot.model.testsuite.TestSuites method*), 189
append_to_environment_variable() (*robot.libraries.OperatingSystem.OperatingSystem method*), 66
append_to_file() (*robot.libraries.OperatingSystem.OperatingSystem method*), 64
append_to_list() (*robot.libraries.Collections.Collections method*), 48
Application (*class in robot.utils.application*), 280
ArgFileParser (*class in robot.utils.argumentparser*), 281
ArgLimitValidator (*class in robot.utils.argumentparser*), 281
args (*robot.errors.ContinueForLoop attribute*), 310
args (*robot.errors.DataError attribute*), 307
args (*robot.errors.ExecutionFailed attribute*), 308
args (*robot.errors.ExecutionFailures attribute*), 308
args (*robot.errors.ExecutionPassed attribute*), 309
args (*robot.errors.ExecutionStatus attribute*), 308
args (*robot.errors.ExitForLoop attribute*), 310
args (*robot.errors.FrameworkError attribute*), 306
args (*robot.errors.HandlerExecutionFailed attribute*), 308
args (*robot.errors.Information attribute*), 307
args (*robot.errors.KeywordError attribute*), 307
args (*robot.errors.PassExecution attribute*), 309
args (*robot.errors.RemoteError attribute*), 311
args (*robot.errors.ReturnFromKeyword attribute*), 310
args (*robot.errors.RobotError attribute*), 306
args (*robot.errors.TimeoutError attribute*), 307
args (*robot.errors.UserKeywordExecutionFailed attribute*), 309
args (*robot.errors.VariableError attribute*), 307
args (*robot.libraries.BuiltIn.RobotNotRunningError attribute*), 46
args (*robot.libraries.Telnet.NoMatchError attribute*), 93
args (*robot.model.keyword.Keyword attribute*), 174
args (*robot.parsing.populators.NoTestsFound attribute*), 213
args (*robot.result.model.Keyword attribute*), 242
args (*robot.running.model.ForLoop attribute*), 267
args (*robot.running.model.Keyword attribute*), 266
argument_names (*robot.running.arguments.argumentspec.ArgumentSpe attribute*), 259
ArgumentCoercer (*class in robot.libraries.Remote*), 75
ArgumentHandler (*class in robot.result.xmlelementhandlers*), 256
ArgumentMapper (*class in robot.running.arguments.argumentmapper*), 258
ArgumentParser (*class in robot.utils.argumentparser*), 280
ArgumentResolver (*class in robot.running.arguments.argumentresolver*), 259
Arguments (*class in robot.parsing.settings*), 216
arguments (*robot.running.userkeywordrunner.EmbeddedArgumentsRunn attribute*), 279
arguments (*robot.running.userkeywordrunner.UserKeywordRunner attribute*), 279
ArgumentsHandler (*class in robot.result.xmlelementhandlers*), 255
ArgumentSpec (*class in robot.running.arguments.argumentspec*),

259
ArgumentValidator (*class in robot.running.arguments.argumentvalidator*), 260
ArgumentValidator (*class in robot.tidy*), 318
as_dict() (*robot.variables.scopes.GlobalVariables method*), 300
as_dict() (*robot.variables.scopes.VariableScopes method*), 300
as_dict() (*robot.variables.store.VariableStore method*), 301
as_dict() (*robot.variables.variables.Variables method*), 303
as_list() (*robot.parsing.comments.Comment method*), 205
as_list() (*robot.parsing.model.ForLoop method*), 212
as_list() (*robot.parsing.model.Step method*), 212
as_list() (*robot.parsing.model.Variable method*), 211
as_list() (*robot.parsing.settings.Arguments method*), 216
as_list() (*robot.parsing.settings.Documentation method*), 214
as_list() (*robot.parsing.settings.Fixture method*), 215
as_list() (*robot.parsing.settings.Library method*), 217
as_list() (*robot.parsing.settings.Metadata method*), 217
as_list() (*robot.parsing.settings.Resource method*), 217
as_list() (*robot.parsing.settings.Return method*), 216
as_list() (*robot.parsing.settings.Setting method*), 214
as_list() (*robot.parsing.settings.Tags method*), 216
as_list() (*robot.parsing.settings.Template method*), 215
as_list() (*robot.parsing.settings.Timeout method*), 215
as_list() (*robot.parsing.settings.Variables method*), 217
aspect() (*robot.libraries.dialogs_py.InputDialog method*), 118
aspect() (*robot.libraries.dialogs_py.MessageDialog method*), 105
aspect() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 144
aspect() (*robot.libraries.dialogs_py.PassFailDialog method*), 157
aspect() (*robot.libraries.dialogs_py.SelectionDialog method*), 131
assert_almost_equal() (*in module robot.utils.asserts*), 283
assert_equal() (*in module robot.utils.asserts*), 283
assert_false() (*in module robot.utils.asserts*), 283
assert_has_content() (*robot.parsing.restsupport.CaptureRobotData method*), 213
assert_none() (*in module robot.utils.asserts*), 283
assert_not_almost_equal() (*in module robot.utils.asserts*), 283
assert_not_equal() (*in module robot.utils.asserts*), 283
assert_not_none() (*in module robot.utils.asserts*), 283
assert_raises() (*in module robot.utils.asserts*), 283
assert_raises_with_msg() (*in module robot.utils.asserts*), 283
assert_true() (*in module robot.utils.asserts*), 283
assign (*robot.model.keyword.Keyword attribute*), 174
assign (*robot.result.model.Keyword attribute*), 242
assign (*robot.running.model.ForLoop attribute*), 267
assign (*robot.running.model.Keyword attribute*), 266
assign() (*robot.variables.assigner.VariableAssigner method*), 297
assigner() (*robot.variables.assigner.VariableAssignment method*), 297
AssignHandler (*class in robot.result.xmlelementhandlers*), 255
AssignmentValidator (*class in robot.variables.assigner*), 297
AssignVarHandler (*class in robot.result.xmlelementhandlers*), 255
attribute_escape() (*in module robot.utils.markuputils*), 290
attributes() (*robot.libraries.dialogs_py.InputDialog method*), 118
attributes() (*robot.libraries.dialogs_py.MessageDialog method*), 105
attributes() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 144
attributes() (*robot.libraries.dialogs_py.PassFailDialog method*), 157
attributes() (*robot.libraries.dialogs_py.SelectionDialog method*), 131

B

bbox() (*robot.libraries.dialogs_py.InputDialog method*), 118
bbox() (*robot.libraries.dialogs_py.MessageDialog method*), 105
bbox() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 144
bbox() (*robot.libraries.dialogs_py.PassFailDialog method*), 157

bbox() (*robot.libraries.dialogs_py.SelectionDialog method*), 131
bell() (*robot.libraries.dialogs_py.InputDialog method*), 118
bell() (*robot.libraries.dialogs_py.MessageDialog method*), 105
bell() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 144
bell() (*robot.libraries.dialogs_py.PassFailDialog method*), 157
bell() (*robot.libraries.dialogs_py.SelectionDialog method*), 131
binary (*robot.libraries.Remote.ArgumentCoercer attribute*), 75
binary_file_writer() (*in module robot.utils.robotio*), 293
bind() (*robot.libraries.dialogs_py.InputDialog method*), 118
bind() (*robot.libraries.dialogs_py.MessageDialog method*), 105
bind() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 144
bind() (*robot.libraries.dialogs_py.PassFailDialog method*), 157
bind() (*robot.libraries.dialogs_py.SelectionDialog method*), 131
bind_all() (*robot.libraries.dialogs_py.InputDialog method*), 119
bind_all() (*robot.libraries.dialogs_py.MessageDialog method*), 105
bind_all() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 145
bind_all() (*robot.libraries.dialogs_py.PassFailDialog method*), 158
bind_all() (*robot.libraries.dialogs_py.SelectionDialog method*), 132
bind_class() (*robot.libraries.dialogs_py.InputDialog method*), 119
bind_class() (*robot.libraries.dialogs_py.MessageDialog method*), 106
bind_class() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 145
bind_class() (*robot.libraries.dialogs_py.PassFailDialog method*), 158
bind_class() (*robot.libraries.dialogs_py.SelectionDialog method*), 132
bindtags() (*robot.libraries.dialogs_py.InputDialog method*), 119
bindtags() (*robot.libraries.dialogs_py.MessageDialog method*), 106
bindtags() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 145
bindtags() (*robot.libraries.dialogs_py.PassFailDialog method*), 158
bindtags() (*robot.libraries.dialogs_py.SelectionDialog method*), 132
bit_length() (*robot.reporting.stringcache.StringIndex method*), 224
br_start() (*robot.parsing.htmlreader.HtmlReader method*), 206
build() (*robot.libdocpkg.javabuilder.JavaDocBuilder method*), 22
build() (*robot.libdocpkg.robotbuilder.LibraryDocBuilder method*), 22
build() (*robot.libdocpkg.robotbuilder.ResourceDocBuilder method*), 23
build() (*robot.libdocpkg.specbuilder.SpecDocBuilder method*), 23
build() (*robot.reporting.jsmodelbuilders.ErrorMessageBuilder method*), 221
build() (*robot.reporting.jsmodelbuilders.ErrorsBuilder method*), 221
build() (*robot.reporting.jsmodelbuilders.KeywordBuilder method*), 221
build() (*robot.reporting.jsmodelbuilders.MessageBuilder method*), 221
build() (*robot.reporting.jsmodelbuilders.StatisticsBuilder method*), 221
build() (*robot.reporting.jsmodelbuilders.SuiteBuilder method*), 221
build() (*robot.reporting.jsmodelbuilders.TestBuilder method*), 221
build() (*robot.result.resultbuilder.ExecutionResultBuilder method*), 248
build() (*robot.running.builder.ResourceFileBuilder method*), 262
build() (*robot.running.builder.TestSuiteBuilder method*), 261
build_from() (*robot.reporting.jsmodelbuilders.JsModelBuilder method*), 221
build_keyword() (*robot.libdocpkg.robotbuilder.KeywordDocBuilder method*), 23
build_keywords() (*robot.libdocpkg.robotbuilder.KeywordDocBuilder method*), 23
build_step() (*robot.running.builder.StepBuilder method*), 262
build_steps() (*robot.running.builder.StepBuilder method*), 262
BuiltIn (*class in robot.libraries.BuiltIn*), 23
by_method_name() (*robot.output.listenerarguments.EndKeywordArgu class method*), 197
by_method_name() (*robot.output.listenerarguments.EndSuiteArgument class method*), 197
by_method_name() (*robot.output.listenerarguments.EndTestArguments class method*), 197
by_method_name() (*robot.output.listenerarguments.ListenerArguments class method*), 196
by_method_name() (*robot.output.listenerarguments.MessageArgument*

class method), 196
by_method_name() (*robot.output.listenerarguments.StartKeywordArguments class method*), 197
by_method_name() (*robot.output.listenerarguments.StartSuiteArguments class method*), 197
by_method_name() (*robot.output.listenerarguments.StartTestArguments class method*), 197
ByNameKeywordRemover (*class in robot.result.keywordremover*), 234
ByPathImporter (*class in robot.utils.importer*), 289
ByTagKeywordRemover (*class in robot.result.keywordremover*), 235

C

cache_only (*robot.output.logger.Logger attribute*), 199
call_method() (*robot.libraries.BuiltIn.BuiltIn method*), 26
called (*robot.output.listenermethods.ListenerMethod attribute*), 197
can_continue() (*robot.errors.ContinueForLoop method*), 310
can_continue() (*robot.errors.ExecutionFailed method*), 308
can_continue() (*robot.errors.ExecutionFailures method*), 308
can_continue() (*robot.errors.ExecutionPassed method*), 309
can_continue() (*robot.errors.ExecutionStatus method*), 307
can_continue() (*robot.errors.ExitForLoop method*), 310
can_continue() (*robot.errors.HandlerExecutionFailed method*), 308
can_continue() (*robot.errors.PassExecution method*), 309
can_continue() (*robot.errors.ReturnFromKeyword method*), 310
can_continue() (*robot.errors.UserKeywordExecutionFailed method*), 309
CaptureRobotData (*class in robot.parsing.restsupport*), 213
catenate() (*robot.libraries.BuiltIn.BuiltIn method*), 26
CDATA_CONTENT_ELEMENTS (*robot.parsing.htmlreader.HtmlReader attribute*), 206
cget() (*robot.libraries.dialogs_py.InputDialog method*), 119
cget() (*robot.libraries.dialogs_py.MessageDialog method*), 106
cget() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 145
cget() (*robot.libraries.dialogs_py.PassFailDialog method*), 158
cget() (*robot.libraries.dialogs_py.SelectionDialog method*), 132
check_for_whole_start_tag() (*robot.parsing.htmlreader.HtmlReader method*), 206
child() (*robot.libraries.XML.Location method*), 104
children (*robot.model.keyword.Keyword attribute*), 175
children (*robot.result.model.Keyword attribute*), 242
children (*robot.running.model.ForLoop attribute*), 267
children (*robot.running.model.Keyword attribute*), 266
ClassDoc() (*in module robot.libdocpkg.javabuilder*), 22
clear() (*robot.model.imports.Imports method*), 173
clear() (*robot.model.itemlist.ItemList method*), 174
clear() (*robot.model.keyword.Keywords method*), 175
clear() (*robot.model.message.Messages method*), 177
clear() (*robot.model.metadata.Metadata method*), 177
clear() (*robot.model.testcase.TestCases method*), 187
clear() (*robot.model.testsuite.TestSuites method*), 189
clear() (*robot.utils.dotdict.DotDict method*), 285
clear() (*robot.utils.normalizing.NormalizedDict method*), 292
clear() (*robot.variables.scopes.GlobalVariables method*), 300
clear() (*robot.variables.store.VariableStore method*), 301
clear() (*robot.variables.variables.Variables method*), 303
clear_cdata_mode() (*robot.parsing.htmlreader.HtmlReader method*), 206
clear_element() (*robot.libraries.XML.XML method*), 103
client() (*robot.libraries.dialogs_py.InputDialog method*), 119
client() (*robot.libraries.dialogs_py.MessageDialog method*), 106
client() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 145
client() (*robot.libraries.dialogs_py.PassFailDialog method*), 158
client() (*robot.libraries.dialogs_py.SelectionDialog method*), 132
clipboard_append() (*robot.libraries.dialogs_py.InputDialog method*), 119
clipboard_append() (*robot.libraries.dialogs_py.MessageDialog*

method), 106
clipboard_append() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 145
clipboard_append() (*robot.libraries.dialogs_py.PassFailDialog method*), 158
clipboard_append() (*robot.libraries.dialogs_py.SelectionDialog method*), 132
clipboard_clear() (*robot.libraries.dialogs_py.InputDialog method*), 119
clipboard_clear() (*robot.libraries.dialogs_py.MessageDialog method*), 106
clipboard_clear() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 145
clipboard_clear() (*robot.libraries.dialogs_py.PassFailDialog method*), 158
clipboard_clear() (*robot.libraries.dialogs_py.SelectionDialog method*), 132
clipboard_get() (*robot.libraries.dialogs_py.InputDialog method*), 119
clipboard_get() (*robot.libraries.dialogs_py.MessageDialog method*), 106
clipboard_get() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 145
clipboard_get() (*robot.libraries.dialogs_py.PassFailDialog method*), 158
clipboard_get() (*robot.libraries.dialogs_py.SelectionDialog method*), 132
close() (*robot.libraries.Remote.TimeoutHTTPSTransport method*), 76
close() (*robot.libraries.Remote.TimeoutHTTPTransport method*), 76
close() (*robot.libraries.Telnet.TelnetConnection method*), 91
close() (*robot.output.filelogger.FileLogger method*), 196
close() (*robot.output.logger.Logger method*), 199
close() (*robot.output.output.Output method*), 201
close() (*robot.output.pyloggingconf.RobotHandler method*), 202
close() (*robot.output.xmllogger.XmlLogger method*), 203
close() (*robot.parsing.htmlreader.HtmlReader method*), 206
close() (*robot.reporting.outputwriter.OutputWriter method*), 222
close() (*robot.utils.application.DefaultLogger method*), 280
close() (*robot.utils.markupwriters.HtmlWriter method*), 290
close() (*robot.utils.markupwriters.NullMarkupWriter method*), 291
close() (*robot.utils.markupwriters.XmlWriter method*), 290
close_all() (*robot.utils.connectioncache.ConnectionCache method*), 285
close_all_connections() (*robot.libraries.Telnet.Telnet method*), 87
close_connection() (*robot.libraries.Telnet.TelnetConnection method*), 89
close_global_library_listeners() (*robot.running.importer.Importer method*), 263
close_streams() (*robot.libraries.Process.ExecutionResult method*), 75
cmdline2list() (*in module robot.utils.argumentparser*), 280
coerce() (*robot.libraries.Remote.ArgumentCoercer method*), 75
Collections (*class in robot.libraries.Collections*), 47
colormapwindows() (*robot.libraries.dialogs_py.InputDialog method*), 119
colormapwindows() (*robot.libraries.dialogs_py.MessageDialog method*), 106
colormapwindows() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 145
colormapwindows() (*robot.libraries.dialogs_py.PassFailDialog method*), 158
colormapwindows() (*robot.libraries.dialogs_py.SelectionDialog method*), 132
colormodel() (*robot.libraries.dialogs_py.InputDialog method*), 119
colormodel() (*robot.libraries.dialogs_py.MessageDialog method*), 106
colormodel() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 145
colormodel() (*robot.libraries.dialogs_py.PassFailDialog method*), 158
colormodel() (*robot.libraries.dialogs_py.SelectionDialog method*), 132
ColumnAligner (*class in robot.writer.aligners*), 303
columnconfigure() (*robot.libraries.dialogs_py.InputDialog method*), 119
columnconfigure()

(*robot.libraries.dialogs_py.MessageDialog method*), 106
columnconfigure() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 145
columnconfigure() (*robot.libraries.dialogs_py.PassFailDialog method*), 158
columnconfigure() (*robot.libraries.dialogs_py.SelectionDialog method*), 132
combine_lists() (*robot.libraries.Collections.Collections method*), 49
combined (*robot.model.stats.TagStat attribute*), 182
combined (*robot.model.tagstatistics.TagStatistics attribute*), 185
CombinedResult (*class in robot.result.executionresult*), 230
CombinedTagStat (*class in robot.model.stats*), 182
command() (*robot.libraries.dialogs_py.InputDialog method*), 120
command() (*robot.libraries.dialogs_py.MessageDialog method*), 106
command() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 146
command() (*robot.libraries.dialogs_py.PassFailDialog method*), 159
command() (*robot.libraries.dialogs_py.SelectionDialog method*), 133
Comment (*class in robot.parsing.comments*), 205
comment() (*robot.libraries.BuiltIn.BuiltIn method*), 26
CommentCache (*class in robot.parsing.comments*), 205
Comments (*class in robot.parsing.comments*), 205
compare() (*robot.libraries.XML.ElementComparator method*), 104
compress_text() (*in module robot.utils.compress*), 284
config() (*robot.libraries.dialogs_py.InputDialog method*), 120
config() (*robot.libraries.dialogs_py.MessageDialog method*), 107
config() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 146
config() (*robot.libraries.dialogs_py.PassFailDialog method*), 159
config() (*robot.libraries.dialogs_py.SelectionDialog method*), 133
configure() (*robot.libraries.dialogs_py.InputDialog method*), 120
configure() (*robot.libraries.dialogs_py.MessageDialog method*), 107
configure() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 146
configure() (*robot.libraries.dialogs_py.PassFailDialog method*), 159
configure() (*robot.libraries.dialogs_py.SelectionDialog method*), 133
configure() (*robot.model.testsuite.TestSuite method*), 188
configure() (*robot.result.executionresult.CombinedResult method*), 231
configure() (*robot.result.executionresult.Result method*), 230
configure() (*robot.result.model.TestSuite method*), 245
configure() (*robot.running.model.TestSuite method*), 269
conjugate() (*robot.reporting.stringcache.StringIndex method*), 224
ConnectionCache (*class in robot.utils.connectioncache*), 284
console() (*in module robot.api.logger*), 14
console() (*in module robot.output.librarylogger*), 196
console() (*robot.libdoc.LibDoc method*), 311
console() (*robot.rebot.Rebot method*), 313
console() (*robot.run.RobotFramework method*), 314
console() (*robot.testdoc.TestDoc method*), 316
console() (*robot.tidy.TidyCommandLine method*), 318
console() (*robot.utils.application.Application method*), 280
console_colors (*robot.conf.settings.RebotSettings attribute*), 18
console_colors (*robot.conf.settings.RobotSettings attribute*), 17
console_decode() (*in module robot.utils.encoding*), 286
console_encode() (*in module robot.utils.encoding*), 286
console_markers (*robot.conf.settings.RobotSettings attribute*), 17
console_output_config (*robot.conf.settings.RebotSettings attribute*), 18
console_output_config (*robot.conf.settings.RobotSettings attribute*), 17
console_type (*robot.conf.settings.RobotSettings attribute*), 17
console_width (*robot.conf.settings.RobotSettings attribute*), 17
ConsoleOutput() (*in module robot.output.console*), 192
ConsoleViewer (*class in robot.libdocpkg.consoleviewer*), 21
consume_with() (*robot.parsing.comments.CommentCache method*), 205
conditions_var() (*in module robot.variables.isvar*), 299
content() (*robot.utils.markupwriters.HtmlWriter*

method), 290
content() (*robot.utils.markupwriters.NullMarkupWriter method*), 291
content() (*robot.utils.markupwriters.XmlWriter method*), 290
continue_for_loop() (*robot.libraries.BuiltIn.BuiltIn method*), 27
continue_for_loop_if() (*robot.libraries.BuiltIn.BuiltIn method*), 27
continue_on_failure (*robot.errors.ContinueForLoop attribute*), 310
continue_on_failure (*robot.errors.ExecutionFailed attribute*), 308
continue_on_failure (*robot.errors.ExecutionFailures attribute*), 308
continue_on_failure (*robot.errors.ExecutionPassed attribute*), 309
continue_on_failure (*robot.errors.ExecutionStatus attribute*), 307
continue_on_failure (*robot.errors.ExitForLoop attribute*), 310
continue_on_failure (*robot.errors.HandlerExecutionFailed attribute*), 308
continue_on_failure (*robot.errors.PassExecution attribute*), 309
continue_on_failure (*robot.errors.ReturnFromKeyword attribute*), 310
continue_on_failure (*robot.errors.UserKeywordExecutionFailed attribute*), 309
ContinueForLoop, 309
convert() (*robot.libdocpkg.htmlwriter.JsonConverter method*), 22
convert() (*robot.testdoc.JsonConverter method*), 316
convert_date() (*in module robot.libraries.DateTime*), 57
convert_time() (*in module robot.libraries.DateTime*), 57
convert_to_binary() (*robot.libraries.BuiltIn.BuiltIn method*), 27
convert_to_boolean() (*robot.libraries.BuiltIn.BuiltIn method*), 27
convert_to_bytes() (*robot.libraries.BuiltIn.BuiltIn method*), 27
convert_to_dictionary() (*robot.libraries.Collections.Collections method*), 49
convert_to_hex() (*robot.libraries.BuiltIn.BuiltIn method*), 27
convert_to_integer() (*robot.libraries.BuiltIn.BuiltIn method*), 28
convert_to_list() (*robot.libraries.Collections.Collections method*), 49
convert_to_lowercase() (*robot.libraries.String.String method*), 79
convert_to_number() (*robot.libraries.BuiltIn.BuiltIn method*), 28
convert_to_octal() (*robot.libraries.BuiltIn.BuiltIn method*), 28
convert_to_string() (*robot.libraries.BuiltIn.BuiltIn method*), 28
convert_to_uppercase() (*robot.libraries.String.String method*), 79
copy() (*robot.model.keyword.Keyword method*), 175
copy() (*robot.model.message.Message method*), 176
copy() (*robot.model.metadata.Metadata method*), 177
copy() (*robot.model.modelobject.ModelObject method*), 178
copy() (*robot.model.testcase.TestCase method*), 186
copy() (*robot.model.testsuite.TestSuite method*), 188
copy() (*robot.output.loggerhelper.Message method*), 200
copy() (*robot.parsing.model.ForLoop method*), 212
copy() (*robot.parsing.model.TestCase method*), 211
copy() (*robot.parsing.model.UserKeyword method*), 211
copy() (*robot.result.model.Keyword method*), 242
copy() (*robot.result.model.Message method*), 241
copy() (*robot.result.model.TestCase method*), 243
copy() (*robot.result.model.TestSuite method*), 246
copy() (*robot.running.model.ForLoop method*), 267
copy() (*robot.running.model.Keyword method*), 266
copy() (*robot.running.model.TestCase method*), 268
copy() (*robot.running.model.TestSuite method*), 270
copy() (*robot.utils.dotdict.DotDict method*), 285
copy() (*robot.utils.normalizing.NormalizedDict method*), 292
copy() (*robot.variables.scopes.GlobalVariables method*), 300
copy() (*robot.variables.variables.Variables method*), 303

copy_dictionary() (*robot.libraries.Collections.Collections method*), 49
copy_directory() (*robot.libraries.OperatingSystem.OperatingSystem method*), 65
copy_element() (*robot.libraries.XML.XML method*), 103
copy_file() (*robot.libraries.OperatingSystem.OperatingSystem method*), 64
copy_files() (*robot.libraries.OperatingSystem.OperatingSystem method*), 65
copy_list() (*robot.libraries.Collections.Collections method*), 49
count_directories_in_directory() (*robot.libraries.OperatingSystem.OperatingSystem method*), 69
count_files_in_directory() (*robot.libraries.OperatingSystem.OperatingSystem method*), 69
count_items_in_directory() (*robot.libraries.OperatingSystem.OperatingSystem method*), 68
count_values_in_list() (*robot.libraries.Collections.Collections method*), 49
create() (*robot.model.imports.Imports method*), 173
create() (*robot.model.itemlist.ItemList method*), 174
create() (*robot.model.keyword.Keywords method*), 175
create() (*robot.model.message.Messages method*), 177
create() (*robot.model.testcase.TestCases method*), 187
create() (*robot.model.testsuite.TestSuites method*), 189
create_binary_file() (*robot.libraries.OperatingSystem.OperatingSystem method*), 64
create_dictionary() (*robot.libraries.BuiltIn.BuiltIn method*), 29
create_directory() (*robot.libraries.OperatingSystem.OperatingSystem method*), 64
create_file() (*robot.libraries.OperatingSystem.OperatingSystem method*), 63
create_link_target() (*robot.reporting.jsbuildingcontext.JsBuildingContext method*), 220
create_list() (*robot.libraries.BuiltIn.BuiltIn method*), 29
create_runner() (*robot.running.handlers.EmbeddedArgumentsHandler method*), 263
create_runner() (*robot.running.handlerstore.HandlerStore method*), 263
create_runner() (*robot.running.usererrorhandler.UserErrorHandler method*), 278
create_runner() (*robot.running.userkeyword.EmbeddedArgumentsHandler method*), 279
create_runner() (*robot.running.userkeyword.UserKeywordHandler method*), 278
createLock() (*robot.output.pyloggingconf.RobotHandler method*), 202
critical (*robot.model.stats.TagStat attribute*), 182
critical (*robot.model.tagstatistics.TagStatistics attribute*), 185
critical (*robot.result.model.TestCase attribute*), 243
critical_failure_occurred() (*robot.running.status.SuiteStatus method*), 276
critical_failure_occurred() (*robot.running.status.TestStatus method*), 276
critical_tags (*robot.conf.settings.RebotSettings attribute*), 18
critical_tags (*robot.conf.settings.RobotSettings attribute*), 17
Criticality (*class in robot.model.criticality*), 171
criticality (*robot.result.model.TestSuite attribute*), 245
CriticalTagStat (*class in robot.model.stats*), 182
CssFileWriter (*class in robot.htmldata.htmlfilewriter*), 19
current (*robot.model.suitestatistics.SuiteStatisticsBuilder attribute*), 183
current (*robot.running.context.ExecutionContexts attribute*), 262
current (*robot.utils.connectioncache.ConnectionCache attribute*), 284
current (*robot.variables.scopes.VariableScopes attribute*), 300
current_index (*robot.utils.connectioncache.ConnectionCache attribute*), 284
current_output (*robot.libraries.Telnet.TerminalEmulator attribute*), 93
cut_long_message() (*in module robot.utils.text*), 296

D

data (*robot.parsing.datarow.DataRow attribute*), 205
DataError, 306
DataExtractor (*class in robot.writer.dataextractor*), 303
DataFileWriter (*class in robot.writer.datafilewriter*), 303
DataRow (*class in robot.parsing.datarow*), 205
debug() (*in module robot.api.logger*), 14
debug() (*in module robot.output.librarylogger*), 196

debug() (*robot.output.filelogger.FileLogger method*), 196
debug() (*robot.output.logger.Logger method*), 199
debug() (*robot.output.loggerhelper.AbstractLogger method*), 200
debug() (*robot.output.output.Output method*), 201
debug() (*robot.parsing.restsupport.CaptureRobotData method*), 213
debug_file (*robot.conf.settings.RobotSettings attribute*), 17
DebugFile() (*in module robot.output.debugfile*), 195
decode_bytes_to_string() (*robot.libraries.String.String method*), 79
dedent() (*robot.parsing.datarow.DataRow method*), 205
deepcopy() (*robot.model.keyword.Keyword method*), 175
deepcopy() (*robot.model.message.Message method*), 177
deepcopy() (*robot.model.modelobject.ModelObject method*), 178
deepcopy() (*robot.model.testcase.TestCase method*), 186
deepcopy() (*robot.model.testsuite.TestSuite method*), 188
deepcopy() (*robot.output.loggerhelper.Message method*), 200
deepcopy() (*robot.result.model.Keyword method*), 242
deepcopy() (*robot.result.model.Message method*), 241
deepcopy() (*robot.result.model.TestCase method*), 244
deepcopy() (*robot.result.model.TestSuite method*), 246
deepcopy() (*robot.running.model.ForLoop method*), 267
deepcopy() (*robot.running.model.Keyword method*), 266
deepcopy() (*robot.running.model.TestCase method*), 268
deepcopy() (*robot.running.model.TestSuite method*), 271
DefaultLogger (*class in robot.utils.application*), 280
DefaultValue (*class in robot.running.arguments.argumentmapper*), 258
deiconify() (*robot.libraries.dialogs_py.InputDialog method*), 120
deiconify() (*robot.libraries.dialogs_py.MessageDialog method*), 107
deiconify() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 146
deiconify() (*robot.libraries.dialogs_py.PassFailDialog method*), 159
deiconify() (*robot.libraries.dialogs_py.SelectionDialog method*), 133
del_env_var() (*in module robot.utils.robotenv*), 293
delayed_logging (*robot.output.logger.Logger attribute*), 199
deletecommand() (*robot.libraries.dialogs_py.InputDialog method*), 120
deletecommand() (*robot.libraries.dialogs_py.MessageDialog method*), 107
deletecommand() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 146
deletecommand() (*robot.libraries.dialogs_py.PassFailDialog method*), 159
deletecommand() (*robot.libraries.dialogs_py.SelectionDialog method*), 133
denominator (*robot.reporting.stringcache.StringIndex attribute*), 225
destroy() (*robot.libraries.dialogs_py.InputDialog method*), 120
destroy() (*robot.libraries.dialogs_py.MessageDialog method*), 107
destroy() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 146
destroy() (*robot.libraries.dialogs_py.PassFailDialog method*), 159
destroy() (*robot.libraries.dialogs_py.SelectionDialog method*), 133
DictDumper (*class in robot.htmldata.jsonwriter*), 20
dictionaries_should_be_equal() (*robot.libraries.Collections.Collections method*), 49
dictionary_should_contain_item() (*robot.libraries.Collections.Collections method*), 49
dictionary_should_contain_key() (*robot.libraries.Collections.Collections method*), 49
dictionary_should_contain_sub_dictionary() (*robot.libraries.Collections.Collections method*), 50
dictionary_should_contain_value() (*robot.libraries.Collections.Collections method*), 50
dictionary_should_not_contain_key() (*robot.libraries.Collections.Collections method*), 50
dictionary_should_not_contain_value() (*robot.libraries.Collections.Collections method*), 50
DictToKwargs (*class in robot.running.arguments.argumentresolver*), 259
DictVariableTableValue (*class in*

robot.variables.tablesetter), 302
directive_error() (*robot.parsing.restsupport.CaptureRobotData method*), 213
directory (*robot.model.imports.Import attribute*), 173
directory (*robot.parsing.model.InitFileSettingTable attribute*), 210
directory (*robot.parsing.model.KeywordTable attribute*), 211
directory (*robot.parsing.model.ResourceFileSettingTable attribute*), 209
directory (*robot.parsing.model.TestCase attribute*), 211
directory (*robot.parsing.model.TestCaseFileSettingTable attribute*), 209
directory (*robot.parsing.model.TestCaseTable attribute*), 210
directory (*robot.parsing.model.UserKeyword attribute*), 211
directory (*robot.parsing.model.VariableTable attribute*), 210
directory (*robot.parsing.settings.Arguments attribute*), 216
directory (*robot.parsing.settings.Documentation attribute*), 214
directory (*robot.parsing.settings.Fixture attribute*), 215
directory (*robot.parsing.settings.Library attribute*), 217
directory (*robot.parsing.settings.Metadata attribute*), 217
directory (*robot.parsing.settings.Resource attribute*), 217
directory (*robot.parsing.settings.Return attribute*), 216
directory (*robot.parsing.settings.Setting attribute*), 214
directory (*robot.parsing.settings.Tags attribute*), 216
directory (*robot.parsing.settings.Template attribute*), 215
directory (*robot.parsing.settings.Timeout attribute*), 215
directory (*robot.parsing.settings.Variables attribute*), 217
directory() (*robot.tidy.Tidy method*), 317
directory_should_be_empty() (*robot.libraries.OperatingSystem.OperatingSystem method*), 63
directory_should_exist() (*robot.libraries.OperatingSystem.OperatingSystem method*), 62
directory_should_not_be_empty() (*robot.libraries.OperatingSystem.OperatingSystem method*), 63
directory_should_not_exist() (*robot.libraries.OperatingSystem.OperatingSystem method*), 62
disable_curdir_processing() (*in module robot.parsing*), 205
disable_library_import_logging() (*robot.output.logger.Logger method*), 199
disable_message_cache() (*robot.output.logger.Logger method*), 199
discard_suite_scope() (*robot.output.listenermethods.LibraryListenerMethods method*), 197
discard_suite_scope() (*robot.output.listeners.LibraryListeners method*), 198
doc (*robot.model.keyword.Keyword attribute*), 174
doc (*robot.model.stats.TagStat attribute*), 182
doc (*robot.model.testcase.TestCase attribute*), 186
doc (*robot.model.testsuite.TestSuite attribute*), 187
doc (*robot.result.model.Keyword attribute*), 242
doc (*robot.result.model.TestCase attribute*), 244
doc (*robot.result.model.TestSuite attribute*), 246
doc (*robot.running.model.ForLoop attribute*), 267
doc (*robot.running.model.Keyword attribute*), 266
doc (*robot.running.model.TestCase attribute*), 269
doc (*robot.running.model.TestSuite attribute*), 271
doc (*robot.running.usererrorhandler.UserErrorHandler attribute*), 278
doc_format (*robot.libdocpkg.model.LibraryDoc attribute*), 22
DocFormatter (*class in robot.libdocpkg.htmlwriter*), 22
DocHandler (*class in robot.result.xmlelementhandlers*), 254
DocToHtml (*class in robot.libdocpkg.htmlwriter*), 22
Documentation (*class in robot.parsing.settings*), 214
DocumentationBuilder() (*in module robot.libdocpkg.builder*), 21
DocumentationCell (*class in robot.writer.htmlformatter*), 306
DocumentationPopulator (*class in robot.parsing.tablepopulators*), 219
dont_continue (*robot.errors.ContinueForLoop attribute*), 310
dont_continue (*robot.errors.ExecutionFailed attribute*), 308
dont_continue (*robot.errors.ExecutionFailures attribute*), 308
dont_continue (*robot.errors.ExecutionPassed attribute*), 309
dont_continue (*robot.errors.ExecutionStatus attribute*), 307
dont_continue (*robot.errors.ExitForLoop attribute*), 310

dont_continue (*robot.errors.HandlerExecutionFailed attribute*), 308
dont_continue (*robot.errors.PassExecution attribute*), 309
dont_continue (*robot.errors.ReturnFromKeyword attribute*), 310
dont_continue (*robot.errors.UserKeywordExecutionFailed attribute*), 309
DosHighlighter (*class in robot.output.console.highlighting*), 194
DotDict (*class in robot.utils.dotdict*), 285
DottedImporter (*class in robot.utils.importer*), 290
DottedOutput (*class in robot.output.console.dotted*), 192
dry_run (*robot.conf.settings.RobotSettings attribute*), 17
dry_run() (*robot.running.librarykeywordrunner.EmbeddedArgumentsRunner method*), 264
dry_run() (*robot.running.librarykeywordrunner.LibraryKeywordRunner method*), 264
dry_run() (*robot.running.librarykeywordrunner.RunKeywordRunner method*), 264
dry_run() (*robot.running.usererrorhandler.UserErrorHandler method*), 278
dry_run() (*robot.running.userkeywordrunner.EmbeddedArgumentsRunner method*), 279
dry_run() (*robot.running.userkeywordrunner.UserKeywordRunner method*), 279
dump() (*robot.htmldata.jsonwriter.DictDumper method*), 20
dump() (*robot.htmldata.jsonwriter.IntegerDumper method*), 20
dump() (*robot.htmldata.jsonwriter.JsonDumper method*), 20
dump() (*robot.htmldata.jsonwriter.MappingDumper method*), 20
dump() (*robot.htmldata.jsonwriter.NoneDumper method*), 20
dump() (*robot.htmldata.jsonwriter.StringDumper method*), 20
dump() (*robot.htmldata.jsonwriter.TupleListDumper method*), 20
dump() (*robot.reporting.stringcache.StringCache method*), 225
DynamicArgumentParser (*class in robot.running.arguments.argumentparser*), 258
DynamicHandler() (*in module robot.running.handlers*), 263

E

earlier_failures (*robot.errors.ContinueForLoop attribute*), 310
earlier_failures (*robot.errors.ExecutionPassed attribute*), 309
earlier_failures (*robot.errors.ExitForLoop attribute*), 310
earlier_failures (*robot.errors.PassExecution attribute*), 309
earlier_failures (*robot.errors.ReturnFromKeyword attribute*), 310
elapsed (*robot.model.stats.Stat attribute*), 181
elapsed (*robot.model.stats.SuiteStat attribute*), 181
elapsed_time_to_string() (*in module robot.utils.robottime*), 295
elapsedtime (*robot.result.model.Keyword attribute*), 242
elapsedtime (*robot.result.model.TestCase attribute*), 243
elapsedtime (*robot.result.model.TestSuite attribute*), 245
element() (*robot.utils.markupwriters.HtmlWriter method*), 290
element() (*robot.utils.markupwriters.NullMarkupWriter method*), 291
element() (*robot.utils.markupwriters.XmlWriter method*), 290
element_attribute_should_be() (*robot.libraries.XML.XML method*), 100
element_attribute_should_match() (*robot.libraries.XML.XML method*), 100
element_should_exist() (*robot.libraries.XML.XML method*), 98
element_should_not_exist() (*robot.libraries.XML.XML method*), 98
element_should_not_have_attribute() (*robot.libraries.XML.XML method*), 100
element_text_should_be() (*robot.libraries.XML.XML method*), 99
element_text_should_match() (*robot.libraries.XML.XML method*), 99
element_to_string() (*robot.libraries.XML.XML method*), 103
ElementComparator (*class in robot.libraries.XML*), 104
ElementFinder (*class in robot.libraries.XML*), 104
elements_should_be_equal() (*robot.libraries.XML.XML method*), 100
elements_should_match() (*robot.libraries.XML.XML method*), 100
EmbeddedArgumentParser (*class in robot.running.arguments.embedded*), 260
EmbeddedArguments (*class in robot.running.arguments.embedded*), 260
EmbeddedArgumentsHandler (*class in robot.running.handlers*), 263
EmbeddedArgumentsHandler (*class in*

robot.running.userkeyword), 279
EmbeddedArgumentsRunner (*class in robot.running.librarykeywordrunner*), 264
EmbeddedArgumentsRunner (*class in robot.running.userkeywordrunner*), 279
emit() (*robot.output.pyloggingconf.RobotHandler method*), 201
empty_cache() (*robot.utils.connectioncache.ConnectionCache method*), 285
empty_directory() (*robot.libraries.OperatingSystem.OperatingSystem method*), 64
empty_row_after() (*robot.writer.formatters.PipeFormatter method*), 305
empty_row_after() (*robot.writer.formatters.TsvFormatter method*), 305
empty_row_after() (*robot.writer.formatters.TxtFormatter method*), 305
empty_row_after() (*robot.writer.htmlformatter.HtmlFormatter method*), 306
EmptyFinder (*class in robot.variables.finders*), 298
EmptySuiteRemover (*class in robot.model.filter*), 171
enable_library_import_logging() (*robot.output.logger.Logger method*), 199
encode_string_to_bytes() (*robot.libraries.String.String method*), 79
encode_threshold (*robot.libraries.Remote.TimeoutHTTPSTransport attribute*), 76
encode_threshold (*robot.libraries.Remote.TimeoutHTTPTransport attribute*), 76
end() (*robot.result.xmlelementhandlers.ArgumentHandler method*), 256
end() (*robot.result.xmlelementhandlers.ArgumentsHandler method*), 256
end() (*robot.result.xmlelementhandlers.AssignHandler method*), 255
end() (*robot.result.xmlelementhandlers.AssignVarHandler method*), 255
end() (*robot.result.xmlelementhandlers.DocHandler method*), 254
end() (*robot.result.xmlelementhandlers.ErrorsHandler method*), 256
end() (*robot.result.xmlelementhandlers.KeywordHandler method*), 254
end() (*robot.result.xmlelementhandlers.KeywordStatusHandler method*), 254
end() (*robot.result.xmlelementhandlers.MessageHandler method*), 254
end() (*robot.result.xmlelementhandlers.MetadataHandler method*), 254
end() (*robot.result.xmlelementhandlers.MetadataItemHandler method*), 255
end() (*robot.result.xmlelementhandlers.RobotHandler method*), 253
end() (*robot.result.xmlelementhandlers.RootHandler method*), 253
end() (*robot.result.xmlelementhandlers.RootSuiteHandler method*), 253
end() (*robot.result.xmlelementhandlers.StatisticsHandler method*), 256
end() (*robot.result.xmlelementhandlers.SuiteHandler method*), 253
end() (*robot.result.xmlelementhandlers.SuiteStatusHandler method*), 254
end() (*robot.result.xmlelementhandlers.TagHandler method*), 255
end() (*robot.result.xmlelementhandlers.TagsHandler method*), 255
end() (*robot.result.xmlelementhandlers.TestCaseHandler method*), 253
end() (*robot.result.xmlelementhandlers.TestStatusHandler method*), 254
end() (*robot.result.xmlelementhandlers.TimeoutHandler method*), 255
end() (*robot.result.xmlelementhandlers.XmlElementHandler method*), 253
end() (*robot.utils.htmlformatters.HeaderFormatter method*), 288
end() (*robot.utils.htmlformatters.ListFormatter method*), 289
end() (*robot.utils.htmlformatters.ParagraphFormatter method*), 288
end() (*robot.utils.htmlformatters.PreformattedFormatter method*), 289
end() (*robot.utils.htmlformatters.RulerFormatter method*), 288
end() (*robot.utils.htmlformatters.TableFormatter method*), 288
end() (*robot.utils.markupwriters.HtmlWriter method*), 290
end() (*robot.utils.markupwriters.NullMarkupWriter method*), 291
end() (*robot.utils.markupwriters.XmlWriter method*), 290
end_errors() (*robot.output.xmllogger.XmlLogger method*), 203
end_errors() (*robot.reporting.outputwriter.OutputWriter method*), 222
end_errors() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 225
end_errors() (*robot.result.visitor.ResultVisitor method*), 252
end_for_loop() (*robot.parsing.model.TestCase*

method), 211
end_for_loop() (*robot.parsing.model.UserKeyword method*), 212
end_keyword() (*robot.conf.gatherfailed.GatherFailedSuites method*), 16
end_keyword() (*robot.conf.gatherfailed.GatherFailedTests method*), 15
end_keyword() (*robot.model.configurer.SuiteConfigurer method*), 170
end_keyword() (*robot.model.filter.EmptySuiteRemover method*), 171
end_keyword() (*robot.model.filter.Filter method*), 172
end_keyword() (*robot.model.modifier.ModelModifier method*), 178
end_keyword() (*robot.model.statistics.StatisticsBuilder method*), 180
end_keyword() (*robot.model.tagsetter.TagSetter method*), 184
end_keyword() (*robot.model.totalstatistics.TotalStatisticsBuilder method*), 190
end_keyword() (*robot.model.visitor.SuiteVisitor method*), 192
end_keyword() (*robot.output.console.dotted.StatusReporter method*), 193
end_keyword() (*robot.output.console.verbose.VerboseOutput method*), 195
end_keyword() (*robot.output.filelogger.FileLogger method*), 196
end_keyword() (*robot.output.logger.Logger method*), 199
end_keyword() (*robot.output.output.Output method*), 201
end_keyword() (*robot.output.xmllogger.XmlLogger method*), 203
end_keyword() (*robot.reporting.outputwriter.OutputWriter method*), 222
end_keyword() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 226
end_keyword() (*robot.result.configurer.SuiteConfigurer method*), 228
end_keyword() (*robot.result.keywordremover.AllKeywordsRemover method*), 232
end_keyword() (*robot.result.keywordremover.ByNameKeywordRemover method*), 234
end_keyword() (*robot.result.keywordremover.ByTagKeywordRemover method*), 235
end_keyword() (*robot.result.keywordremover.ForLoopItemsRemover method*), 236
end_keyword() (*robot.result.keywordremover.PassedKeywordRemover method*), 233
end_keyword() (*robot.result.keywordremover.WaitUntilKeywordSucceedsRemover method*), 237
end_keyword() (*robot.result.keywordremover.WarningAndErrorFinder method*), 238
end_keyword() (*robot.result.merger.Merger method*), 239
end_keyword() (*robot.result.messagefilter.MessageFilter method*), 239
end_keyword() (*robot.result.resultbuilder.RemoveKeywords method*), 248
end_keyword() (*robot.result.suiteteardownfailed.SuiteTeardownFailed method*), 250
end_keyword() (*robot.result.suiteteardownfailed.SuiteTeardownFailureHandler method*), 249
end_keyword() (*robot.result.visitor.ResultVisitor method*), 252
end_keyword() (*robot.running.randomizer.Randomizer method*), 274
end_keyword() (*robot.running.runner.Runner method*), 275
end_keyword() (*robot.variables.scopes.SetVariables method*), 301
end_keyword() (*robot.variables.scopes.VariableScopes method*), 300
end_loggers (*robot.output.logger.Logger attribute*), 198
end_message() (*robot.conf.gatherfailed.GatherFailedSuites method*), 16
end_message() (*robot.conf.gatherfailed.GatherFailedTests method*), 15
end_message() (*robot.model.configurer.SuiteConfigurer method*), 170
end_message() (*robot.model.filter.EmptySuiteRemover method*), 171
end_message() (*robot.model.filter.Filter method*), 172
end_message() (*robot.model.modifier.ModelModifier method*), 178
end_message() (*robot.model.statistics.StatisticsBuilder method*), 180
end_message() (*robot.model.tagsetter.TagSetter method*), 184
end_message() (*robot.model.totalstatistics.TotalStatisticsBuilder method*), 190
end_message() (*robot.model.visitor.SuiteVisitor method*), 192
end_message() (*robot.output.console.dotted.StatusReporter method*), 193
end_message() (*robot.output.xmllogger.XmlLogger method*), 204
end_message() (*robot.reporting.outputwriter.OutputWriter method*), 223
end_message() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 226
end_message() (*robot.result.configurer.SuiteConfigurer method*), 228
end_message() (*robot.result.keywordremover.AllKeywordsRemover*

method), 232
end_message() (*robot.result.keywordremover.ByNameKeywordRemover method*), 234
end_message() (*robot.result.keywordremover.ByTagKeywordRemover method*), 235
end_message() (*robot.result.keywordremover.ForLoopItemsRemover method*), 236
end_message() (*robot.result.keywordremover.PassedKeywordRemover method*), 233
end_message() (*robot.result.keywordremover.WaitUntilKeywordSucceedsRemover method*), 237
end_message() (*robot.result.keywordremover.WarningAndErrorFinder method*), 238
end_message() (*robot.result.merger.Merger method*), 239
end_message() (*robot.result.messagefilter.MessageFilter method*), 240
end_message() (*robot.result.resultbuilder.RemoveKeywords method*), 248
end_message() (*robot.result.suiteteardownfailed.SuiteTeardownFailed method*), 250
end_message() (*robot.result.suiteteardownfailed.SuiteTeardownFailureHandler method*), 249
end_message() (*robot.result.visitor.ResultVisitor method*), 252
end_message() (*robot.running.randomizer.Randomizer method*), 274
end_message() (*robot.running.runner.Runner method*), 275
end_result() (*robot.output.xmllogger.XmlLogger method*), 204
end_result() (*robot.reporting.outputwriter.OutputWriter method*), 222
end_result() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 225
end_result() (*robot.result.visitor.ResultVisitor method*), 251
end_splitting() (*robot.reporting.jsbuildingcontext.JsBuildingContext method*), 220
end_stat() (*robot.output.xmllogger.XmlLogger method*), 204
end_stat() (*robot.reporting.outputwriter.OutputWriter method*), 223
end_stat() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 226
end_stat() (*robot.result.visitor.ResultVisitor method*), 252
end_statistics() (*robot.output.xmllogger.XmlLogger method*), 203
end_statistics() (*robot.reporting.outputwriter.OutputWriter method*), 223
end_statistics() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 226
end_statistics() (*robot.result.visitor.ResultVisitor method*), 251
end_suite() (*robot.conf.gatherfailed.GatherFailedSuites method*), 16
end_suite() (*robot.conf.gatherfailed.GatherFailedTests method*), 15
end_suite() (*robot.model.configurer.SuiteConfigurer method*), 170
end_suite() (*robot.model.filter.EmptySuiteRemover method*), 171
end_suite() (*robot.model.filter.Filter method*), 172
end_suite() (*robot.model.modifier.ModelModifier method*), 178
end_suite() (*robot.model.statistics.StatisticsBuilder method*), 180
end_suite() (*robot.model.suitestatistics.SuiteStatisticsBuilder method*), 183
end_suite() (*robot.model.tagsetter.TagSetter method*), 184
end_suite() (*robot.model.totalstatistics.TotalStatisticsBuilder method*), 190
end_suite() (*robot.model.visitor.SuiteVisitor method*), 191
end_suite() (*robot.output.console.dotted.DottedOutput method*), 192
end_suite() (*robot.output.console.dotted.StatusReporter method*), 193
end_suite() (*robot.output.console.verbose.VerboseOutput method*), 195
end_suite() (*robot.output.filelogger.FileLogger method*), 196
end_suite() (*robot.output.logger.Logger method*), 199
end_suite() (*robot.output.output.Output method*), 201
end_suite() (*robot.output.xmllogger.XmlLogger method*), 203
end_suite() (*robot.reporting.outputwriter.OutputWriter method*), 223
end_suite() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 225
end_suite() (*robot.result.configurer.SuiteConfigurer method*), 228
end_suite() (*robot.result.keywordremover.AllKeywordsRemover method*), 232
end_suite() (*robot.result.keywordremover.ByNameKeywordRemover method*), 234
end_suite() (*robot.result.keywordremover.ByTagKeywordRemover method*), 235
end_suite() (*robot.result.keywordremover.ForLoopItemsRemover method*), 236
end_suite() (*robot.result.keywordremover.PassedKeywordRemover method*), 233
end_suite() (*robot.result.keywordremover.WaitUntilKeywordSucceedsR method*), 237

end_suite() (*robot.result.keywordremover.WarningAndErrorFinder method*), 238
end_suite() (*robot.result.merger.Merger method*), 239
end_suite() (*robot.result.messagefilter.MessageFilter method*), 240
end_suite() (*robot.result.resultbuilder.RemoveKeywords method*), 248
end_suite() (*robot.result.suiteteardownfailed.SuiteTeardownFailed method*), 250
end_suite() (*robot.result.suiteteardownfailed.SuiteTeardownFailureHandler method*), 249
end_suite() (*robot.result.visitor.ResultVisitor method*), 252
end_suite() (*robot.running.context.ExecutionContexts method*), 262
end_suite() (*robot.running.libraryscopes.GlobalScope method*), 265
end_suite() (*robot.running.libraryscopes.TestCaseScope method*), 265
end_suite() (*robot.running.libraryscopes.TestSuiteScope method*), 265
end_suite() (*robot.running.namespace.Namespace method*), 273
end_suite() (*robot.running.randomizer.Randomizer method*), 274
end_suite() (*robot.running.runner.Runner method*), 275
end_suite() (*robot.variables.scopes.SetVariables method*), 301
end_suite() (*robot.variables.scopes.VariableScopes method*), 300
end_suite_statistics() (*robot.output.xmllogger.XmlLogger method*), 203
end_suite_statistics() (*robot.reporting.outputwriter.OutputWriter method*), 223
end_suite_statistics() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 226
end_suite_statistics() (*robot.result.visitor.ResultVisitor method*), 251
end_tag_statistics() (*robot.output.xmllogger.XmlLogger method*), 203
end_tag_statistics() (*robot.reporting.outputwriter.OutputWriter method*), 223
end_tag_statistics() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 226
end_tag_statistics() (*robot.result.visitor.ResultVisitor method*), 251
end_test() (*robot.conf.gatherfailed.GatherFailedSuites method*), 16
end_test() (*robot.conf.gatherfailed.GatherFailedTests method*), 15
end_test() (*robot.model.configurer.SuiteConfigurer method*), 170
end_test() (*robot.model.filter.EmptySuiteRemover method*), 171
end_test() (*robot.model.filter.Filter method*), 172
end_test() (*robot.model.modifier.ModelModifier method*), 178
end_test() (*robot.model.statistics.StatisticsBuilder method*), 180
end_test() (*robot.model.tagsetter.TagSetter method*), 184
end_test() (*robot.model.totalstatistics.TotalStatisticsBuilder method*), 190
end_test() (*robot.model.visitor.SuiteVisitor method*), 191
end_test() (*robot.output.console.dotted.DottedOutput method*), 192
end_test() (*robot.output.console.dotted.StatusReporter method*), 193
end_test() (*robot.output.console.verbose.VerboseOutput method*), 195
end_test() (*robot.output.filelogger.FileLogger method*), 196
end_test() (*robot.output.logger.Logger method*), 199
end_test() (*robot.output.output.Output method*), 201
end_test() (*robot.output.xmllogger.XmlLogger method*), 203
end_test() (*robot.reporting.outputwriter.OutputWriter method*), 223
end_test() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 226
end_test() (*robot.result.configurer.SuiteConfigurer method*), 228
end_test() (*robot.result.keywordremover.AllKeywordsRemover method*), 232
end_test() (*robot.result.keywordremover.ByNameKeywordRemover method*), 234
end_test() (*robot.result.keywordremover.ByTagKeywordRemover method*), 235
end_test() (*robot.result.keywordremover.ForLoopItemsRemover method*), 236
end_test() (*robot.result.keywordremover.PassedKeywordRemover method*), 233
end_test() (*robot.result.keywordremover.WaitUntilKeywordSucceedsRe method*), 237
end_test() (*robot.result.keywordremover.WarningAndErrorFinder method*), 238
end_test() (*robot.result.merger.Merger method*), 239

end_test() (*robot.result.messagefilter.MessageFilter method*), 240
end_test() (*robot.result.resultbuilder.RemoveKeywords method*), 248
end_test() (*robot.result.suiteteardownfailed.SuiteTeardownFailed method*), 250
end_test() (*robot.result.suiteteardownfailed.SuiteTeardownFailureHandler method*), 249
end_test() (*robot.result.visitor.ResultVisitor method*), 252
end_test() (*robot.running.libraryscopes.GlobalScope method*), 265
end_test() (*robot.running.libraryscopes.TestCaseScope method*), 265
end_test() (*robot.running.libraryscopes.TestSuiteScope method*), 265
end_test() (*robot.running.namespace.Namespace method*), 273
end_test() (*robot.running.randomizer.Randomizer method*), 274
end_test() (*robot.running.runner.Runner method*), 275
end_test() (*robot.variables.scopes.SetVariables method*), 301
end_test() (*robot.variables.scopes.VariableScopes method*), 300
end_total_statistics() (*robot.output.xmllogger.XmlLogger method*), 203
end_total_statistics() (*robot.reporting.outputwriter.OutputWriter method*), 223
end_total_statistics() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 226
end_total_statistics() (*robot.result.visitor.ResultVisitor method*), 251
end_user_keyword() (*robot.running.namespace.Namespace method*), 273
EndKeywordArguments (*class in robot.output.listenerarguments*), 197
EndSuiteArguments (*class in robot.output.listenerarguments*), 197
EndTestArguments (*class in robot.output.listenerarguments*), 197
endtime (*robot.result.model.Keyword attribute*), 242
endtime (*robot.result.model.TestCase attribute*), 243
endtime (*robot.result.model.TestSuite attribute*), 244
entitydefs (*robot.parsing.htmlreader.HtmlReader attribute*), 206
environment_variable_should_be_set() (*robot.libraries.OperatingSystem.OperatingSystem method*), 66
environment_variable_should_not_be_set() (*robot.libraries.OperatingSystem.OperatingSystem method*), 66
EnvironmentFinder (*class in robot.variables.finders*), 299
eof() (*robot.parsing.populators.FromFilePopulator method*), 213
eq() (*in module robot.utils.match*), 291
error() (*in module robot.api.logger*), 14
error() (*in module robot.output.librarylogger*), 196
error() (*robot.output.console.highlighting.HighlightingStream method*), 194
error() (*robot.output.console.verbose.VerboseWriter method*), 195
error() (*robot.output.filelogger.FileLogger method*), 196
error() (*robot.output.logger.Logger method*), 199
error() (*robot.output.loggerhelper.AbstractLogger method*), 200
error() (*robot.output.output.Output method*), 201
error() (*robot.parsing.htmlreader.HtmlReader method*), 206
error() (*robot.parsing.restsupport.CaptureRobotData method*), 213
error() (*robot.utils.application.DefaultLogger method*), 280
error_occurred() (*robot.running.status.Exit method*), 276
error_occurred() (*robot.running.status.SuiteStatus method*), 276
error_occurred() (*robot.running.status.TestStatus method*), 276
ErrorDetails() (*in module robot.utils.error*), 286
ErrorMessageBuilder (*class in robot.reporting.jsmodelbuilders*), 221
errors (*robot.result.executionresult.Result attribute*), 229
ErrorsBuilder (*class in robot.reporting.jsmodelbuilders*), 221
ErrorsHandler (*class in robot.result.xmlelementhandlers*), 256
escape() (*in module robot.utils.escaping*), 287
EscapeFinder (*class in robot.utils.escaping*), 287
ETSource (*class in robot.utils.etreewrapper*), 287
evaluate() (*robot.libraries.BuiltIn.BuiltIn method*), 29
evaluate_xpath() (*robot.libraries.XML.XML method*), 104
event_add() (*robot.libraries.dialogs_py.InputDialog method*), 120
event_add() (*robot.libraries.dialogs_py.MessageDialog method*), 107
event_add() (*robot.libraries.dialogs_py.MultipleSelectionDialog*

method), 146
event_add() (*robot.libraries.dialogs_py.PassFailDialog method*), 159
event_add() (*robot.libraries.dialogs_py.SelectionDialog method*), 133
event_delete() (*robot.libraries.dialogs_py.InputDialog method*), 120
event_delete() (*robot.libraries.dialogs_py.MessageDialog method*), 107
event_delete() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 146
event_delete() (*robot.libraries.dialogs_py.PassFailDialog method*), 159
event_delete() (*robot.libraries.dialogs_py.SelectionDialog method*), 133
event_generate() (*robot.libraries.dialogs_py.InputDialog method*), 120
event_generate() (*robot.libraries.dialogs_py.MessageDialog method*), 107
event_generate() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 146
event_generate() (*robot.libraries.dialogs_py.PassFailDialog method*), 159
event_generate() (*robot.libraries.dialogs_py.SelectionDialog method*), 133
event_info() (*robot.libraries.dialogs_py.InputDialog method*), 120
event_info() (*robot.libraries.dialogs_py.MessageDialog method*), 107
event_info() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 146
event_info() (*robot.libraries.dialogs_py.PassFailDialog method*), 159
event_info() (*robot.libraries.dialogs_py.SelectionDialog method*), 133
exclude_tags (*robot.model.filter.Filter attribute*), 172
execute() (*robot.libdoc.LibDoc method*), 311
execute() (*robot.rebot.Rebot method*), 313
execute() (*robot.run.RobotFramework method*), 314
execute() (*robot.running.timeouts.posix.Timeout method*), 261
execute() (*robot.running.timeouts.windows.Timeout method*), 261
execute() (*robot.testdoc.TestDoc method*), 316
execute() (*robot.tidy.TidyCommandLine method*), 318
execute() (*robot.utils.application.Application method*), 280
execute_cli() (*robot.libdoc.LibDoc method*), 311
execute_cli() (*robot.rebot.Rebot method*), 313
execute_cli() (*robot.run.RobotFramework method*), 314
execute_cli() (*robot.testdoc.TestDoc method*), 316
execute_cli() (*robot.tidy.TidyCommandLine method*), 318
execute_cli() (*robot.utils.application.Application method*), 280
execute_command() (*robot.libraries.Telnet.TelnetConnection method*), 91
execute_manual_step() (*in module robot.libraries.Dialogs*), 59
ExecutionContexts (*class in robot.running.context*), 262
ExecutionErrors (*class in robot.result.executionerrors*), 229
ExecutionFailed, 308
ExecutionFailures, 308
ExecutionPassed, 309
ExecutionResult (*class in robot.libraries.Process*), 74
ExecutionResult() (*in module robot.result.resultbuilder*), 247
ExecutionResultBuilder (*class in robot.result.resultbuilder*), 248
ExecutionStatus, 307
Exit (*class in robot.running.status*), 276
exit_for_loop() (*robot.libraries.BuiltIn.BuiltIn method*), 29
exit_for_loop_if() (*robot.libraries.BuiltIn.BuiltIn method*), 29
exit_on_error (*robot.conf.settings.RobotSettings attribute*), 17
exit_on_error_message (*robot.running.status.TestMessage attribute*), 277
exit_on_failure (*robot.conf.settings.RobotSettings attribute*), 17
exit_on_failure_message (*robot.running.status.TestMessage attribute*), 276
exit_on_fatal_message (*robot.running.status.TestMessage attribute*), 276
ExitForLoop, 310
expect() (*robot.libraries.Telnet.TelnetConnection method*), 91
extend() (*robot.model.imports.Imports method*), 173
extend() (*robot.model.itemlist.ItemList method*), 174
extend() (*robot.model.keyword.Keywords method*), 175
extend() (*robot.model.message.Messages method*), 177
extend() (*robot.model.testcase.TestCases method*), 187
extend() (*robot.model.testsuite.TestSuites method*), 189

ExtendedFinder (*class in robot.variables.finders*), 299
extension (*robot.conf.settings.RobotSettings attribute*), 17

F

fail() (*in module robot.utils.asserts*), 283
fail() (*robot.libraries.BuiltIn.BuiltIn method*), 30
fail() (*robot.output.filelogger.FileLogger method*), 196
fail() (*robot.output.logger.Logger method*), 199
fail() (*robot.output.loggerhelper.AbstractLogger method*), 200
fail() (*robot.output.output.Output method*), 201
failed (*robot.model.stats.Stat attribute*), 181
Failure (*class in robot.running.status*), 276
failure_occurred() (*robot.running.status.Exit method*), 276
failures (*robot.running.status.SuiteStatus attribute*), 276
failures (*robot.running.status.TestStatus attribute*), 276
fatal_error() (*robot.libraries.BuiltIn.BuiltIn method*), 30
feed() (*robot.libraries.Telnet.TerminalEmulator method*), 93
feed() (*robot.parsing.htmlreader.HtmlReader method*), 207
fetch_from_left() (*robot.libraries.String.String method*), 82
fetch_from_right() (*robot.libraries.String.String method*), 82
file() (*robot.tidy.Tidy method*), 317
file_should_be_empty() (*robot.libraries.OperatingSystem.OperatingSystem method*), 63
file_should_exist() (*robot.libraries.OperatingSystem.OperatingSystem method*), 62
file_should_not_be_empty() (*robot.libraries.OperatingSystem.OperatingSystem method*), 63
file_should_not_exist() (*robot.libraries.OperatingSystem.OperatingSystem method*), 62
file_writer() (*in module robot.utils.robotio*), 293
FileLogger (*class in robot.output.filelogger*), 195
fileno() (*robot.libraries.Telnet.TelnetConnection method*), 91
FileWriter() (*in module robot.writer.filewriters*), 305
fill_named() (*robot.running.arguments.argumentmapper.KeywordCallTemplate method*), 258
fill_positional() (*robot.running.arguments.argumentmapper.KeywordCallTemplate method*), 258
fill_rawq() (*robot.libraries.Telnet.TelnetConnection method*), 91
Filter (*class in robot.model.filter*), 172
filter() (*robot.model.testsuite.TestSuite method*), 188
filter() (*robot.output.pyloggingconf.RobotHandler method*), 202
filter() (*robot.result.model.TestSuite method*), 246
filter() (*robot.running.model.TestSuite method*), 271
filter_messages() (*robot.result.model.TestSuite method*), 245
final_argument_whitespace (*robot.parsing.restsupport.CaptureRobotData attribute*), 213
find() (*robot.variables.finders.EmptyFinder class method*), 299
find() (*robot.variables.finders.EnvironmentFinder method*), 299
find() (*robot.variables.finders.ExtendedFinder method*), 299
find() (*robot.variables.finders.NumberFinder method*), 298
find() (*robot.variables.finders.StoredFinder method*), 298
find() (*robot.variables.finders.VariableFinder method*), 298
find_all() (*robot.libraries.XML.ElementFinder method*), 104
find_file() (*in module robot.utils.robotpath*), 294
find_recommendations() (*robot.utils.recommendations.RecommendationFinder method*), 293
FirstColumnAligner (*class in robot.writer.aligners*), 303
Fixture (*class in robot.parsing.settings*), 215
flatten_keywords (*robot.conf.settings.RebotSettings attribute*), 18
flatten_keywords (*robot.conf.settings.RobotSettings attribute*), 17
FlattenByNameMatcher (*class in robot.result.flattenkeywordmatcher*), 232
FlattenByTagMatcher (*class in robot.result.flattenkeywordmatcher*), 232
FlattenByTypeMatcher (*class in robot.result.flattenkeywordmatcher*), 232
flavor (*robot.running.model.ForLoop attribute*), 267
flavors (*robot.parsing.model.ForLoop attribute*), 212
flush() (*robot.output.console.highlighting.HighlightingStream method*), 194
flush() (*robot.output.pyloggingconf.RobotHandler method*), 202
focus() (*robot.libraries.dialogs_py.InputDialog*

method), 120
focus() (*robot.libraries.dialogs_py.MessageDialog method*), 107
focus() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 146
focus() (*robot.libraries.dialogs_py.PassFailDialog method*), 159
focus() (*robot.libraries.dialogs_py.SelectionDialog method*), 133
focus_displayof() (*robot.libraries.dialogs_py.InputDialog method*), 120
focus_displayof() (*robot.libraries.dialogs_py.MessageDialog method*), 107
focus_displayof() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 146
focus_displayof() (*robot.libraries.dialogs_py.PassFailDialog method*), 159
focus_displayof() (*robot.libraries.dialogs_py.SelectionDialog method*), 133
focus_force() (*robot.libraries.dialogs_py.InputDialog method*), 120
focus_force() (*robot.libraries.dialogs_py.MessageDialog method*), 107
focus_force() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 146
focus_force() (*robot.libraries.dialogs_py.PassFailDialog method*), 159
focus_force() (*robot.libraries.dialogs_py.SelectionDialog method*), 133
focus_get() (*robot.libraries.dialogs_py.InputDialog method*), 120
focus_get() (*robot.libraries.dialogs_py.MessageDialog method*), 107
focus_get() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 146
focus_get() (*robot.libraries.dialogs_py.PassFailDialog method*), 159
focus_get() (*robot.libraries.dialogs_py.SelectionDialog method*), 133
focus_lastfor() (*robot.libraries.dialogs_py.InputDialog method*), 121
focus_lastfor() (*robot.libraries.dialogs_py.MessageDialog method*), 107
focus_lastfor() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 147
focus_lastfor() (*robot.libraries.dialogs_py.PassFailDialog method*), 160
focus_lastfor() (*robot.libraries.dialogs_py.SelectionDialog method*), 134
focus_set() (*robot.libraries.dialogs_py.InputDialog method*), 121
focus_set() (*robot.libraries.dialogs_py.MessageDialog method*), 108
focus_set() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 147
focus_set() (*robot.libraries.dialogs_py.PassFailDialog method*), 160
focus_set() (*robot.libraries.dialogs_py.SelectionDialog method*), 134
focusmodel() (*robot.libraries.dialogs_py.InputDialog method*), 121
focusmodel() (*robot.libraries.dialogs_py.MessageDialog method*), 108
focusmodel() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 147
focusmodel() (*robot.libraries.dialogs_py.PassFailDialog method*), 160
focusmodel() (*robot.libraries.dialogs_py.SelectionDialog method*), 134
FOR_ITEM_TYPE (*robot.model.keyword.Keyword attribute*), 174
FOR_ITEM_TYPE (*robot.result.model.Keyword attribute*), 242
FOR_ITEM_TYPE (*robot.running.model.ForLoop attribute*), 267
FOR_ITEM_TYPE (*robot.running.model.Keyword attribute*), 265
FOR_LOOP_TYPE (*robot.model.keyword.Keyword attribute*), 174
FOR_LOOP_TYPE (*robot.result.model.Keyword attribute*), 242
FOR_LOOP_TYPE (*robot.running.model.ForLoop attribute*), 267
FOR_LOOP_TYPE (*robot.running.model.Keyword attribute*), 266
ForInEnumerateRunner (*class in robot.running.steprunner*), 277
ForInRangeRunner (*class in robot.running.steprunner*), 277
ForInRunner (*class in robot.running.steprunner*), 277
ForInZipRunner (*class in robot.running.steprunner*), 277
ForLoop (*class in robot.parsing.model*), 212
ForLoop (*class in robot.running.model*), 267
ForLoopItemsRemover (*class in robot.result.keywordremover*), 235
ForLoopPopulator (*class in robot.parsing.tablepopulators*), 219
format() (*robot.output.pyloggingconf.RobotHandler method*), 202
format() (*robot.tidy.ArgumentValidator method*), 318
format() (*robot.utils.htmlformatters.HeaderFormatter method*), 288

format() (*robot.utils.htmlformatters.HtmlFormatter method*), 288
format() (*robot.utils.htmlformatters.LineFormatter method*), 288
format() (*robot.utils.htmlformatters.ListFormatter method*), 289
format() (*robot.utils.htmlformatters.ParagraphFormatter method*), 288
format() (*robot.utils.htmlformatters.PreformattedFormatter method*), 289
format() (*robot.utils.htmlformatters.RulerFormatter method*), 288
format() (*robot.utils.htmlformatters.TableFormatter method*), 288
format() (*robot.utils.unic.PrettyRepr method*), 297
format_assign_message() (*in module robot.utils.text*), 296
format_header() (*robot.writer.formatters.PipeFormatter method*), 305
format_header() (*robot.writer.formatters.TsvFormatter method*), 305
format_header() (*robot.writer.formatters.TxtFormatter method*), 305
format_header() (*robot.writer.htmlformatter.HtmlFormatter method*), 305
format_line() (*robot.utils.htmlformatters.HeaderFormatter method*), 288
format_line() (*robot.utils.htmlformatters.RulerFormatter method*), 288
format_link() (*robot.utils.htmlformatters.LinkFormatter method*), 287
format_recommendations() (*robot.running.namespace.KeywordRecommendationFinder static method*), 273
format_recommendations() (*robot.utils.recommendations.RecommendationFinder static method*), 293
format_string() (*robot.libraries.String.String method*), 79
format_table() (*robot.writer.formatters.PipeFormatter method*), 305
format_table() (*robot.writer.formatters.TsvFormatter method*), 305
format_table() (*robot.writer.formatters.TxtFormatter method*), 305
format_table() (*robot.writer.htmlformatter.HtmlFormatter method*), 306
format_time() (*in module robot.utils.robottime*), 294
format_url() (*robot.utils.htmlformatters.LinkFormatter method*), 287
ForRunner() (*in module robot.running.steprunner*), 277
frame() (*robot.libraries.dialogs_py.InputDialog method*), 121
frame() (*robot.libraries.dialogs_py.MessageDialog method*), 108
frame() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 147
frame() (*robot.libraries.dialogs_py.PassFailDialog method*), 160
frame() (*robot.libraries.dialogs_py.SelectionDialog method*), 134
FrameworkError, 306
frange() (*in module robot.utils.frange*), 287
FromDirectoryPopulator (*class in robot.parsing.populators*), 213
FromFilePopulator (*class in robot.parsing.populators*), 213
fromkeys() (*robot.utils.dotdict.DotDict class method*), 285
full_message (*robot.result.model.TestSuite attribute*), 245

G

gather_failed_suites() (*in module robot.conf.gatherfailed*), 17
gather_failed_tests() (*in module robot.conf.gatherfailed*), 17
GatherFailedSuites (*class in robot.conf.gatherfailed*), 16
GatherFailedTests (*class in robot.conf.gatherfailed*), 15
generate_random_string() (*robot.libraries.String.String method*), 82
GeneratorWriter (*class in robot.htmldata.htmlfilewriter*), 19
geometry() (*robot.libraries.dialogs_py.InputDialog method*), 121
geometry() (*robot.libraries.dialogs_py.MessageDialog method*), 108
geometry() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 147
geometry() (*robot.libraries.dialogs_py.PassFailDialog method*), 160
geometry() (*robot.libraries.dialogs_py.SelectionDialog method*), 134
get() (*robot.model.metadata.Metadata method*), 177
get() (*robot.utils.dotdict.DotDict method*), 285
get() (*robot.utils.normalizing.NormalizedDict method*), 292
get_arguments() (*robot.output.listenerarguments.EndKeywordArgume method*), 197
get_arguments() (*robot.output.listenerarguments.EndSuiteArguments method*), 197
get_arguments() (*robot.output.listenerarguments.EndTestArguments method*), 197
get_arguments() (*robot.output.listenerarguments.ListenerArguments method*), 196

get_arguments() (*robot.output.listenerarguments.MessageArguments method*), 197
get_arguments() (*robot.output.listenerarguments.StartKeywordArguments method*), 197
get_arguments() (*robot.output.listenerarguments.StartSuiteArguments method*), 197
get_arguments() (*robot.output.listenerarguments.StartTestArguments method*), 197
get_attributes() (*robot.model.stats.CombinedTagStat method*), 182
get_attributes() (*robot.model.stats.CriticalTagStat method*), 182
get_attributes() (*robot.model.stats.Stat method*), 181
get_attributes() (*robot.model.stats.SuiteStat method*), 181
get_attributes() (*robot.model.stats.TagStat method*), 182
get_attributes() (*robot.model.stats.TotalStat method*), 181
get_binary_file() (*robot.libraries.OperatingSystem.OperatingSystem method*), 62
get_char_width() (*in module robot.utils.charwidth*), 284
get_child_elements() (*robot.libraries.XML.XML method*), 98
get_child_handler() (*robot.result.xmlelementhandlers.ArgumentHandler method*), 256
get_child_handler() (*robot.result.xmlelementhandlers.ArgumentsHandler method*), 256
get_child_handler() (*robot.result.xmlelementhandlers.AssignHandler method*), 255
get_child_handler() (*robot.result.xmlelementhandlers.AssignVarHandler method*), 255
get_child_handler() (*robot.result.xmlelementhandlers.DocHandler method*), 254
get_child_handler() (*robot.result.xmlelementhandlers.ErrorsHandler method*), 256
get_child_handler() (*robot.result.xmlelementhandlers.KeywordHandler method*), 254
get_child_handler() (*robot.result.xmlelementhandlers.KeywordStatusHandler method*), 254
get_child_handler() (*robot.result.xmlelementhandlers.MessageHandler method*), 254
get_child_handler() (*robot.result.xmlelementhandlers.MetadataHandler method*), 254
get_child_handler() (*robot.result.xmlelementhandlers.MetadataItemHandler method*), 255
get_child_handler() (*robot.result.xmlelementhandlers.RobotHandler method*), 253
get_child_handler() (*robot.result.xmlelementhandlers.RootHandler method*), 253
get_child_handler() (*robot.result.xmlelementhandlers.RootSuiteHandler method*), 253
get_child_handler() (*robot.result.xmlelementhandlers.StatisticsHandler method*), 256
get_child_handler() (*robot.result.xmlelementhandlers.SuiteHandler method*), 253
get_child_handler() (*robot.result.xmlelementhandlers.SuiteStatusHandler method*), 254
get_child_handler() (*robot.result.xmlelementhandlers.TagHandler method*), 255
get_child_handler() (*robot.result.xmlelementhandlers.TagsHandler method*), 255
get_child_handler() (*robot.result.xmlelementhandlers.TestCaseHandler method*), 253
get_child_handler() (*robot.result.xmlelementhandlers.TestStatusHandler method*), 254
get_child_handler() (*robot.result.xmlelementhandlers.TimeoutHandler method*), 255
get_combined_stats() (*robot.model.tagstatistics.TagStatInfo method*), 185
get_command() (*robot.libraries.Process.ProcessConfiguration method*), 75
get_connection() (*robot.utils.connectioncache.ConnectionCache method*), 285
get_console_encoding() (*in module robot.utils.encodingsniffer*), 286
get_console_length() (*in module robot.utils.text*), 296
get_count() (*robot.libraries.BuiltIn.BuiltIn method*), 30
get_critical_stats() (*robot.model.tagstatistics.TagStatInfo method*),

185
get_current_date() (*in module robot.libraries.DateTime*), 56
get_data() (*robot.parsing.restsupport.RobotDataStorage method*), 214
get_dictionary_items() (*robot.libraries.Collections.Collections method*), 50
get_dictionary_keys() (*robot.libraries.Collections.Collections method*), 50
get_dictionary_values() (*robot.libraries.Collections.Collections method*), 50
get_doc() (*robot.model.tagstatistics.TagStatInfo method*), 185
get_elapsed_time() (*in module robot.utils.robottime*), 295
get_element() (*robot.libraries.XML.XML method*), 97
get_element_attribute() (*robot.libraries.XML.XML method*), 99
get_element_attributes() (*robot.libraries.XML.XML method*), 99
get_element_count() (*robot.libraries.XML.XML method*), 98
get_element_text() (*robot.libraries.XML.XML method*), 98
get_elements() (*robot.libraries.XML.XML method*), 98
get_elements_texts() (*robot.libraries.XML.XML method*), 99
get_env_var() (*in module robot.utils.robotenv*), 293
get_env_vars() (*in module robot.utils.robotenv*), 293
get_environment_variable() (*robot.libraries.OperatingSystem.OperatingSystem method*), 65
get_environment_variables() (*robot.libraries.OperatingSystem.OperatingSystem method*), 66
get_error_details() (*in module robot.utils.error*), 286
get_error_message() (*in module robot.utils.error*), 286
get_errors() (*robot.errors.ContinueForLoop method*), 310
get_errors() (*robot.errors.ExecutionFailed method*), 308
get_errors() (*robot.errors.ExecutionFailures method*), 308
get_errors() (*robot.errors.ExecutionPassed method*), 309
get_errors() (*robot.errors.ExecutionStatus method*), 308
get_errors() (*robot.errors.ExitForLoop method*), 310
get_errors() (*robot.errors.HandlerExecutionFailed method*), 308
get_errors() (*robot.errors.PassExecution method*), 309
get_errors() (*robot.errors.ReturnFromKeyword method*), 310
get_errors() (*robot.errors.UserKeywordExecutionFailed method*), 309
get_file() (*robot.libraries.OperatingSystem.OperatingSystem method*), 61
get_file_size() (*robot.libraries.OperatingSystem.OperatingSystem method*), 68
get_from_dictionary() (*robot.libraries.Collections.Collections method*), 50
get_from_list() (*robot.libraries.Collections.Collections method*), 51
get_full_version() (*in module robot.version*), 318
get_host_info() (*robot.libraries.Remote.TimeoutHTTPSTransport method*), 76
get_host_info() (*robot.libraries.Remote.TimeoutHTTPTransport method*), 76
get_index_from_list() (*robot.libraries.Collections.Collections method*), 51
get_interpreter() (*in module robot.version*), 318
get_java_properties() (*in module robot.variables.finders*), 298
get_java_property() (*in module robot.variables.finders*), 298
get_keyword_arguments() (*robot.libraries.Remote.Remote method*), 75
get_keyword_arguments() (*robot.libraries.Remote.XmlRpcRemoteClient method*), 75
get_keyword_documentation() (*robot.libraries.Remote.Remote method*), 75
get_keyword_documentation() (*robot.libraries.Remote.XmlRpcRemoteClient method*), 76
get_keyword_names() (*robot.libraries.Remote.Remote method*), 75
get_keyword_names() (*robot.libraries.Remote.XmlRpcRemoteClient method*), 75
get_keyword_names() (*robot.libraries.Reserved.Reserved method*), 77

get_keyword_names() (*robot.libraries.Telnet.Telnet method*), 87
get_keyword_tags() (*robot.libraries.Remote.Remote method*), 75
get_keyword_tags() (*robot.libraries.Remote.XmlRpcRemoteClient method*), 75
get_keyword_types() (*robot.libraries.Remote.Remote method*), 75
get_keyword_types() (*robot.libraries.Remote.XmlRpcRemoteClient method*), 75
get_length() (*robot.libraries.BuiltIn.BuiltIn method*), 30
get_library() (*robot.running.namespace.KeywordStore method*), 273
get_library_instance() (*robot.libraries.BuiltIn.BuiltIn method*), 30
get_library_instance() (*robot.running.namespace.Namespace method*), 273
get_library_instances() (*robot.running.namespace.Namespace method*), 273
get_line() (*robot.libraries.String.String method*), 80
get_line_count() (*robot.libraries.String.String method*), 79
get_lines_containing_string() (*robot.libraries.String.String method*), 80
get_lines_matching_pattern() (*robot.libraries.String.String method*), 80
get_lines_matching_regexp() (*robot.libraries.String.String method*), 80
get_link() (*robot.model.tagstatistics.TagStatLink method*), 185
get_link_path() (*in module robot.utils.robotpath*), 294
get_links() (*robot.model.tagstatistics.TagStatInfo method*), 185
get_match_count() (*robot.libraries.Collections.Collections method*), 48
get_matches() (*robot.libraries.Collections.Collections method*), 48
get_message() (*robot.running.timeouts.KeywordTimeout method*), 260
get_message() (*robot.running.timeouts.TestTimeout method*), 260
get_modified_time() (*robot.libraries.OperatingSystem.OperatingSystem method*), 67
get_name() (*robot.output.pyloggingconf.RobotHandler method*), 202
get_process_id() (*robot.libraries.Process.Process method*), 74
get_process_object() (*robot.libraries.Process.Process method*), 74
get_process_result() (*robot.libraries.Process.Process method*), 74
get_rebot_settings() (*robot.conf.settings.RobotSettings method*), 17
get_regexp_matches() (*robot.libraries.String.String method*), 81
get_replaced_variable() (*robot.variables.splitter.VariableSplitter method*), 301
get_runner() (*robot.running.namespace.KeywordStore method*), 273
get_runner() (*robot.running.namespace.Namespace method*), 273
get_selection_from_user() (*in module robot.libraries.Dialogs*), 59
get_selections_from_user() (*in module robot.libraries.Dialogs*), 59
get_setter() (*robot.parsing.model.InitFileSettingTable method*), 210
get_setter() (*robot.parsing.model.ResourceFileSettingTable method*), 209
get_setter() (*robot.parsing.model.TestCase method*), 211
get_setter() (*robot.parsing.model.TestCaseFileSettingTable method*), 209
get_setter() (*robot.parsing.model.UserKeyword method*), 212
get_slice_from_list() (*robot.libraries.Collections.Collections method*), 51
get_socket() (*robot.libraries.Telnet.TelnetConnection method*), 91
get_starttag_text() (*robot.parsing.htmlreader.HtmlReader method*), 207
get_stat() (*robot.model.tagstatistics.TagStatInfo method*), 185
get_substring() (*robot.libraries.String.String method*), 82
get_system_encoding() (*in module robot.utils.encodingsniffer*), 286
get_test_values() (*robot.running.defaults.TestDefaults method*), 262
get_time() (*in module robot.utils.robottime*), 294

get_time() (*robot.libraries.BuiltIn.BuiltIn method*), 30
get_timestamp() (*in module robot.utils.robottime*), 295
get_timestamp() (*robot.utils.robottime.TimestampCache method*), 295
get_value_from_user() (*in module robot.libraries.Dialogs*), 59
get_variable_value() (*robot.libraries.BuiltIn.BuiltIn method*), 31
get_variables() (*robot.libraries.BuiltIn.BuiltIn method*), 31
get_version() (*in module robot.version*), 318
getboolean() (*robot.libraries.dialogs_py.InputDialog method*), 121
getboolean() (*robot.libraries.dialogs_py.MessageDialog method*), 108
getboolean() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 147
getboolean() (*robot.libraries.dialogs_py.PassFailDialog method*), 160
getboolean() (*robot.libraries.dialogs_py.SelectionDialog method*), 134
getdoc() (*in module robot.utils.text*), 296
getdouble (*robot.libraries.dialogs_py.InputDialog attribute*), 121
getdouble (*robot.libraries.dialogs_py.MessageDialog attribute*), 108
getdouble (*robot.libraries.dialogs_py.MultipleSelectionDialog attribute*), 147
getdouble (*robot.libraries.dialogs_py.PassFailDialog attribute*), 160
getdouble (*robot.libraries.dialogs_py.SelectionDialog attribute*), 134
getfullargspec() (*in module robot.running.arguments.argumentparser*), 258
getint (*robot.libraries.dialogs_py.InputDialog attribute*), 121
getint (*robot.libraries.dialogs_py.MessageDialog attribute*), 108
getint (*robot.libraries.dialogs_py.MultipleSelectionDialog attribute*), 147
getint (*robot.libraries.dialogs_py.PassFailDialog attribute*), 160
getint (*robot.libraries.dialogs_py.SelectionDialog attribute*), 134
GetKeywordArguments (*class in robot.running.dynamicmethods*), 262
GetKeywordDocumentation (*class in robot.running.dynamicmethods*), 262
GetKeywordNames (*class in robot.running.dynamicmethods*), 262
GetKeywordTags (*class in robot.running.dynamicmethods*), 263
GetKeywordTypes (*class in robot.running.dynamicmethods*), 263
getparser() (*robot.libraries.Remote.TimeoutHTTPSTransport method*), 76
getparser() (*robot.libraries.Remote.TimeoutHTTPTransport method*), 76
getpos() (*robot.parsing.htmlreader.HtmlReader method*), 207
getshortdoc() (*in module robot.utils.text*), 296
getvar() (*robot.libraries.dialogs_py.InputDialog method*), 121
getvar() (*robot.libraries.dialogs_py.MessageDialog method*), 108
getvar() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 147
getvar() (*robot.libraries.dialogs_py.PassFailDialog method*), 160
getvar() (*robot.libraries.dialogs_py.SelectionDialog method*), 134
GlobalScope (*class in robot.running.libraryscopes*), 264
GlobalVariables (*class in robot.variables.scopes*), 300
goahead() (*robot.parsing.htmlreader.HtmlReader method*), 207
grab_current() (*robot.libraries.dialogs_py.InputDialog method*), 121
grab_current() (*robot.libraries.dialogs_py.MessageDialog method*), 108
grab_current() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 147
grab_current() (*robot.libraries.dialogs_py.PassFailDialog method*), 160
grab_current() (*robot.libraries.dialogs_py.SelectionDialog method*), 134
grab_release() (*robot.libraries.dialogs_py.InputDialog method*), 121
grab_release() (*robot.libraries.dialogs_py.MessageDialog method*), 108
grab_release() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 147
grab_release() (*robot.libraries.dialogs_py.PassFailDialog method*), 160
grab_release() (*robot.libraries.dialogs_py.SelectionDialog method*), 134
grab_set() (*robot.libraries.dialogs_py.InputDialog method*), 121
grab_set() (*robot.libraries.dialogs_py.MessageDialog method*), 108
grab_set() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 147
grab_set() (*robot.libraries.dialogs_py.PassFailDialog*

method), 160
grab_set() (*robot.libraries.dialogs_py.SelectionDialog method*), 134
grab_set_global() (*robot.libraries.dialogs_py.InputDialog method*), 121
grab_set_global() (*robot.libraries.dialogs_py.MessageDialog method*), 108
grab_set_global() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 147
grab_set_global() (*robot.libraries.dialogs_py.PassFailDialog method*), 160
grab_set_global() (*robot.libraries.dialogs_py.SelectionDialog method*), 134
grab_status() (*robot.libraries.dialogs_py.InputDialog method*), 121
grab_status() (*robot.libraries.dialogs_py.MessageDialog method*), 108
grab_status() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 147
grab_status() (*robot.libraries.dialogs_py.PassFailDialog method*), 160
grab_status() (*robot.libraries.dialogs_py.SelectionDialog method*), 134
green() (*robot.output.console.highlighting.AnsiHighlighter method*), 194
green() (*robot.output.console.highlighting.DosHighlighter method*), 194
green() (*robot.output.console.highlighting.NoHighlighting method*), 194
grep_file() (*robot.libraries.OperatingSystem.OperatingSystem method*), 62
grid() (*robot.libraries.dialogs_py.InputDialog method*), 121
grid() (*robot.libraries.dialogs_py.MessageDialog method*), 108
grid() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 147
grid() (*robot.libraries.dialogs_py.PassFailDialog method*), 160
grid() (*robot.libraries.dialogs_py.SelectionDialog method*), 134
grid_bbox() (*robot.libraries.dialogs_py.InputDialog method*), 121
grid_bbox() (*robot.libraries.dialogs_py.MessageDialog method*), 108
grid_bbox() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 147
grid_bbox() (*robot.libraries.dialogs_py.PassFailDialog method*), 160
grid_bbox() (*robot.libraries.dialogs_py.SelectionDialog method*), 134
grid_columnconfigure() (*robot.libraries.dialogs_py.InputDialog method*), 122
grid_columnconfigure() (*robot.libraries.dialogs_py.MessageDialog method*), 108
grid_columnconfigure() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 148
grid_columnconfigure() (*robot.libraries.dialogs_py.PassFailDialog method*), 161
grid_columnconfigure() (*robot.libraries.dialogs_py.SelectionDialog method*), 135
grid_location() (*robot.libraries.dialogs_py.InputDialog method*), 122
grid_location() (*robot.libraries.dialogs_py.MessageDialog method*), 109
grid_location() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 148
grid_location() (*robot.libraries.dialogs_py.PassFailDialog method*), 161
grid_location() (*robot.libraries.dialogs_py.SelectionDialog method*), 135
grid_propagate() (*robot.libraries.dialogs_py.InputDialog method*), 122
grid_propagate() (*robot.libraries.dialogs_py.MessageDialog method*), 109
grid_propagate() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 148
grid_propagate() (*robot.libraries.dialogs_py.PassFailDialog method*), 161
grid_propagate() (*robot.libraries.dialogs_py.SelectionDialog method*), 135
grid_rowconfigure() (*robot.libraries.dialogs_py.InputDialog method*), 122
grid_rowconfigure() (*robot.libraries.dialogs_py.MessageDialog method*), 109
grid_rowconfigure() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 148
grid_rowconfigure() (*robot.libraries.dialogs_py.PassFailDialog method*), 161
grid_rowconfigure() (*robot.libraries.dialogs_py.SelectionDialog method*), 135
grid_size() (*robot.libraries.dialogs_py.InputDialog method*), 122

grid_size() (*robot.libraries.dialogs_py.MessageDialog method*), 109
grid_size() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 148
grid_size() (*robot.libraries.dialogs_py.PassFailDialog method*), 161
grid_size() (*robot.libraries.dialogs_py.SelectionDialog method*), 135
grid_slaves() (*robot.libraries.dialogs_py.InputDialog method*), 122
grid_slaves() (*robot.libraries.dialogs_py.MessageDialog method*), 109
grid_slaves() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 148
grid_slaves() (*robot.libraries.dialogs_py.PassFailDialog method*), 161
grid_slaves() (*robot.libraries.dialogs_py.SelectionDialog method*), 135
group() (*robot.libraries.dialogs_py.InputDialog method*), 122
group() (*robot.libraries.dialogs_py.MessageDialog method*), 109
group() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 148
group() (*robot.libraries.dialogs_py.PassFailDialog method*), 161
group() (*robot.libraries.dialogs_py.SelectionDialog method*), 135

H

handle() (*robot.output.pyloggingconf.RobotHandler method*), 202
handle() (*robot.running.arguments.argumentresolver.DictToKwargs method*), 259
handle_charref() (*robot.parsing.htmlreader.HtmlReader method*), 206
handle_comment() (*robot.parsing.htmlreader.HtmlReader method*), 207
handle_data() (*robot.parsing.htmlreader.HtmlReader method*), 206
handle_decl() (*robot.parsing.htmlreader.HtmlReader method*), 207
handle_endtag() (*robot.parsing.htmlreader.HtmlReader method*), 206
handle_entityref() (*robot.parsing.htmlreader.HtmlReader method*), 206
handle_imports() (*robot.running.namespace.Namespace method*), 272
handle_pi() (*robot.parsing.htmlreader.HtmlReader method*), 206
handle_startendtag() (*robot.parsing.htmlreader.HtmlReader method*), 207
handle_starttag() (*robot.parsing.htmlreader.HtmlReader method*), 206
handle_suite_teardown_failures() (*robot.result.executionresult.CombinedResult method*), 231
handle_suite_teardown_failures() (*robot.result.executionresult.Result method*), 230
handle_suite_teardown_failures() (*robot.result.model.TestSuite method*), 246
handleError() (*robot.output.pyloggingconf.RobotHandler method*), 202
Handler() (*in module robot.running.handlers*), 263
HandlerExecutionFailed, 308
HandlerStore (*class in robot.running.handlerstore*), 263
handles() (*robot.htmldata.htmlfilewriter.CssFileWriter method*), 19
handles() (*robot.htmldata.htmlfilewriter.GeneratorWriter method*), 19
handles() (*robot.htmldata.htmlfilewriter.JsFileWriter method*), 19
handles() (*robot.htmldata.htmlfilewriter.LineWriter method*), 19
handles() (*robot.htmldata.htmlfilewriter.ModelWriter method*), 19
handles() (*robot.htmldata.jsonwriter.DictDumper method*), 20
handles() (*robot.htmldata.jsonwriter.IntegerDumper method*), 20
handles() (*robot.htmldata.jsonwriter.MappingDumper method*), 20
handles() (*robot.htmldata.jsonwriter.NoneDumper method*), 20
handles() (*robot.htmldata.jsonwriter.StringDumper method*), 20
handles() (*robot.htmldata.jsonwriter.TupleListDumper method*), 20
handles() (*robot.libdocpkg.consoleviewer.ConsoleViewer class method*), 21
handles() (*robot.libdocpkg.htmlwriter.LibdocModelWriter method*), 21
handles() (*robot.reporting.logreportwriters.RobotModelWriter method*), 222
handles() (*robot.testdoc.TestdocModelWriter method*), 316
handles() (*robot.utils.htmlformatters.HeaderFormatter method*), 288
handles() (*robot.utils.htmlformatters.LineFormatter method*), 288
handles() (*robot.utils.htmlformatters.ListFormatter method*), 289
handles() (*robot.utils.htmlformatters.ParagraphFormatter*

method), 288
handles() (*robot.utils.htmlformatters.PreformattedFormatter method*), 289
handles() (*robot.utils.htmlformatters.RulerFormatter method*), 288
handles() (*robot.utils.htmlformatters.TableFormatter method*), 288
handles() (*robot.utils.importer.ByPathImporter method*), 289
handles() (*robot.utils.importer.DottedImporter method*), 290
handles() (*robot.utils.importer.NonDottedImporter method*), 290
has_content (*robot.parsing.restsupport.CaptureRobotData attribute*), 214
has_data() (*robot.parsing.model.Variable method*), 211
has_data() (*robot.parsing.restsupport.RobotDataStorage method*), 214
has_key() (*robot.utils.dotdict.DotDict method*), 285
has_tests() (*robot.parsing.model.TestCaseFile method*), 208
has_tests() (*robot.parsing.model.TestDataDirectory method*), 209
head (*robot.parsing.datarow.DataRow attribute*), 205
header (*robot.parsing.model.InitFileSettingTable attribute*), 210
header (*robot.parsing.model.KeywordTable attribute*), 211
header (*robot.parsing.model.ResourceFileSettingTable attribute*), 209
header (*robot.parsing.model.TestCaseFileSettingTable attribute*), 209
header (*robot.parsing.model.TestCaseTable attribute*), 210
header (*robot.parsing.model.VariableTable attribute*), 210
HeaderCell (*class in robot.writer.htmlformatter*), 306
HeaderFormatter (*class in robot.utils.htmlformatters*), 288
highlight() (*robot.output.console.highlighting.HighlightingStream method*), 194
Highlighter() (*in module robot.output.console.highlighting*), 194
HighlightingStream (*class in robot.output.console.highlighting*), 194
html (*robot.model.message.Message attribute*), 176
html (*robot.output.loggerhelper.Message attribute*), 200
html (*robot.result.model.Message attribute*), 241
html() (*robot.libdocpkg.htmlwriter.DocFormatter method*), 22
html() (*robot.reporting.jsbuildingcontext.JsBuildingContext method*), 220
html_column_count (*robot.writer.datafilewriter.WritingContext attribute*), 304
html_escape() (*in module robot.utils.markuputils*), 290
html_format (*robot.writer.datafilewriter.WritingContext attribute*), 304
html_format() (*in module robot.utils.markuputils*), 290
html_message (*robot.model.message.Message attribute*), 176
html_message (*robot.output.loggerhelper.Message attribute*), 200
html_message (*robot.result.model.Message attribute*), 241
HtmlCell (*class in robot.writer.htmlformatter*), 306
HtmlFileWriter (*class in robot.htmldata.htmlfilewriter*), 19
HtmlFileWriter (*class in robot.writer.filewriters*), 305
HtmlFormatter (*class in robot.utils.htmlformatters*), 288
HtmlFormatter (*class in robot.writer.htmlformatter*), 305
HtmlReader (*class in robot.parsing.htmlreader*), 206
HtmlTemplate (*class in robot.htmldata.normaltemplate*), 20
HtmlWriter (*class in robot.utils.markupwriters*), 290

I

iconbitmap() (*robot.libraries.dialogs_py.InputDialog method*), 122
iconbitmap() (*robot.libraries.dialogs_py.MessageDialog method*), 109
iconbitmap() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 148
iconbitmap() (*robot.libraries.dialogs_py.PassFailDialog method*), 161
iconbitmap() (*robot.libraries.dialogs_py.SelectionDialog method*), 135
iconify() (*robot.libraries.dialogs_py.InputDialog method*), 122
iconify() (*robot.libraries.dialogs_py.MessageDialog method*), 109
iconify() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 148
iconify() (*robot.libraries.dialogs_py.PassFailDialog method*), 161
iconify() (*robot.libraries.dialogs_py.SelectionDialog method*), 135
iconmask() (*robot.libraries.dialogs_py.InputDialog method*), 122
iconmask() (*robot.libraries.dialogs_py.MessageDialog method*), 109

iconmask() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 148
iconmask() (*robot.libraries.dialogs_py.PassFailDialog method*), 161
iconmask() (*robot.libraries.dialogs_py.SelectionDialog method*), 135
iconname() (*robot.libraries.dialogs_py.InputDialog method*), 122
iconname() (*robot.libraries.dialogs_py.MessageDialog method*), 109
iconname() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 148
iconname() (*robot.libraries.dialogs_py.PassFailDialog method*), 161
iconname() (*robot.libraries.dialogs_py.SelectionDialog method*), 135
iconposition() (*robot.libraries.dialogs_py.InputDialog method*), 122
iconposition() (*robot.libraries.dialogs_py.MessageDialog method*), 109
iconposition() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 148
iconposition() (*robot.libraries.dialogs_py.PassFailDialog method*), 161
iconposition() (*robot.libraries.dialogs_py.SelectionDialog method*), 135
iconwindow() (*robot.libraries.dialogs_py.InputDialog method*), 122
iconwindow() (*robot.libraries.dialogs_py.MessageDialog method*), 109
iconwindow() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 148
iconwindow() (*robot.libraries.dialogs_py.PassFailDialog method*), 161
iconwindow() (*robot.libraries.dialogs_py.SelectionDialog method*), 135
id (*robot.model.keyword.Keyword attribute*), 175
id (*robot.model.stats.SuiteStat attribute*), 181
id (*robot.model.testcase.TestCase attribute*), 186
id (*robot.model.testsuite.TestSuite attribute*), 188
id (*robot.result.model.Keyword attribute*), 243
id (*robot.result.model.TestCase attribute*), 244
id (*robot.result.model.TestSuite attribute*), 246
id (*robot.running.model.ForLoop attribute*), 267
id (*robot.running.model.Keyword attribute*), 266
id (*robot.running.model.TestCase attribute*), 269
id (*robot.running.model.TestSuite attribute*), 271
identifiers (*robot.variables.finders.EmptyFinder attribute*), 299
identifiers (*robot.variables.finders.EnvironmentFinder attribute*), 299
identifiers (*robot.variables.finders.ExtendedFinder attribute*), 299
identifiers (*robot.variables.finders.NumberFinder attribute*), 298
identifiers (*robot.variables.finders.StoredFinder attribute*), 298
IGNORE (*robot.parsing.htmlreader.HtmlReader attribute*), 206
ignored_dirs (*robot.parsing.populators.FromDirectoryPopulator attribute*), 213
ignored_prefixes (*robot.parsing.populators.FromDirectoryPopulator attribute*), 213
imag (*robot.reporting.stringcache.StringIndex attribute*), 225
image_names() (*robot.libraries.dialogs_py.InputDialog method*), 122
image_names() (*robot.libraries.dialogs_py.MessageDialog method*), 109
image_names() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 148
image_names() (*robot.libraries.dialogs_py.PassFailDialog method*), 161
image_names() (*robot.libraries.dialogs_py.SelectionDialog method*), 135
image_types() (*robot.libraries.dialogs_py.InputDialog method*), 122
image_types() (*robot.libraries.dialogs_py.MessageDialog method*), 109
image_types() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 148
image_types() (*robot.libraries.dialogs_py.PassFailDialog method*), 161
image_types() (*robot.libraries.dialogs_py.SelectionDialog method*), 135
Import (*class in robot.model.imports*), 173
import_() (*robot.utils.importer.ByPathImporter method*), 289
import_() (*robot.utils.importer.DottedImporter method*), 290
import_() (*robot.utils.importer.NonDottedImporter method*), 290
import_class_or_module() (*robot.utils.importer.Importer method*), 289
import_class_or_module_by_path() (*robot.utils.importer.Importer method*), 289
import_library() (*robot.libraries.BuiltIn.BuiltIn method*), 31
import_library() (*robot.running.importer.Importer method*), 263
import_library() (*robot.running.namespace.Namespace method*), 273
import_listeners() (*robot.output.listeners.ListenerProxy class method*), 198
import_resource() (*robot.libraries.BuiltIn.BuiltIn method*), 32
import_resource()

(*robot.running.importer.Importer method*), 263
import_resource() (*robot.running.namespace.Namespace method*), 272
import_variables() (*robot.libraries.BuiltIn.BuiltIn method*), 32
import_variables() (*robot.running.namespace.Namespace method*), 272
import_variables() (*robot.variables.filesetter.PythonImporter method*), 298
import_variables() (*robot.variables.filesetter.YamlImporter method*), 298
ImportCache (*class in robot.running.importer*), 263
imported() (*robot.output.listeners.LibraryListeners method*), 198
imported() (*robot.output.listeners.Listeners method*), 198
imported() (*robot.output.logger.Logger method*), 199
Importer (*class in robot.running.importer*), 263
Importer (*class in robot.utils.importer*), 289
ImportList (*class in robot.parsing.settings*), 218
Imports (*class in robot.model.imports*), 173
imports (*robot.parsing.model.ResourceFile attribute*), 208
imports (*robot.parsing.model.TestCaseFile attribute*), 208
imports (*robot.parsing.model.TestDataDirectory attribute*), 209
imports (*robot.running.model.ResourceFile attribute*), 272
include_suites (*robot.model.filter.Filter attribute*), 172
include_tags (*robot.model.filter.Filter attribute*), 172
include_tests (*robot.model.filter.Filter attribute*), 172
index() (*robot.model.imports.Imports method*), 173
index() (*robot.model.itemlist.ItemList method*), 174
index() (*robot.model.keyword.Keywords method*), 175
index() (*robot.model.message.Messages method*), 177
index() (*robot.model.testcase.TestCases method*), 187
index() (*robot.model.testsuite.TestSuites method*), 189
info (*robot.model.stats.CombinedTagStat attribute*), 182
info (*robot.model.stats.CriticalTagStat attribute*), 183
info (*robot.model.stats.TagStat attribute*), 182
info() (*in module robot.api.logger*), 14
info() (*in module robot.output.librarylogger*), 196
info() (*robot.output.console.verbose.VerboseWriter method*), 195
info() (*robot.output.filelogger.FileLogger method*), 196
info() (*robot.output.logger.Logger method*), 199
info() (*robot.output.loggerhelper.AbstractLogger method*), 200
info() (*robot.output.output.Output method*), 201
info() (*robot.parsing.restsupport.CaptureRobotData method*), 214
info() (*robot.utils.application.DefaultLogger method*), 280
Information, 307
InitFileSettingTable (*class in robot.parsing.model*), 210
InitHandler() (*in module robot.running.handlers*), 263
INITIAL (*robot.parsing.htmlreader.HtmlReader attribute*), 206
inplace() (*robot.tidy.Tidy method*), 317
InputDialog (*class in robot.libraries.dialogs_py*), 117
insert() (*robot.model.imports.Imports method*), 173
insert() (*robot.model.itemlist.ItemList method*), 174
insert() (*robot.model.keyword.Keywords method*), 176
insert() (*robot.model.message.Messages method*), 177
insert() (*robot.model.testcase.TestCases method*), 187
insert() (*robot.model.testsuite.TestSuites method*), 189
insert_into_list() (*robot.libraries.Collections.Collections method*), 51
IntegerDumper (*class in robot.htmldata.jsonwriter*), 20
interact() (*robot.libraries.Telnet.TelnetConnection method*), 91
INTERNAL_UPDATE_FREQUENCY (*robot.libraries.Telnet.TelnetConnection attribute*), 88
invalidate_import_caches() (*in module robot.utils.importer*), 289
InvalidForRunner (*class in robot.running.steprunner*), 278
is_active() (*robot.parsing.settings.Fixture method*), 215
is_active() (*robot.parsing.settings.Template method*), 215
is_bytes() (*in module robot.utils.robottypes2*), 296
is_comment() (*robot.parsing.model.ForLoop method*), 212
is_comment() (*robot.parsing.model.Step method*), 212
is_comment() (*robot.parsing.settings.Fixture*

method), 215
is_commented() (*robot.parsing.datarow.DataRow method*), 206
is_continuing() (*robot.parsing.datarow.DataRow method*), 206
is_dict_like() (*in module robot.utils.robottypes2*), 296
is_dict_var() (*in module robot.variables.isvar*), 299
is_dict_variable() (*robot.variables.splitter.VariableSplitter method*), 301
is_falsy() (*in module robot.utils.robottypes*), 295
is_for_loop() (*robot.parsing.model.ForLoop method*), 212
is_for_loop() (*robot.parsing.model.Step method*), 212
is_for_loop() (*robot.parsing.model.Variable method*), 211
is_for_loop() (*robot.parsing.settings.Arguments method*), 216
is_for_loop() (*robot.parsing.settings.Documentation method*), 214
is_for_loop() (*robot.parsing.settings.Fixture method*), 215
is_for_loop() (*robot.parsing.settings.Library method*), 217
is_for_loop() (*robot.parsing.settings.Metadata method*), 217
is_for_loop() (*robot.parsing.settings.Resource method*), 217
is_for_loop() (*robot.parsing.settings.Return method*), 216
is_for_loop() (*robot.parsing.settings.Setting method*), 214
is_for_loop() (*robot.parsing.settings.Tags method*), 216
is_for_loop() (*robot.parsing.settings.Template method*), 215
is_for_loop() (*robot.parsing.settings.Timeout method*), 215
is_for_loop() (*robot.parsing.settings.Variables method*), 217
is_global (*robot.running.libraryscopes.GlobalScope attribute*), 264
is_global (*robot.running.libraryscopes.TestCaseScope attribute*), 265
is_global (*robot.running.libraryscopes.TestSuiteScope attribute*), 265
is_indented() (*robot.parsing.datarow.DataRow method*), 206
is_integer() (*in module robot.utils.robottypes2*), 295
is_java_init() (*in module robot.utils.robotinspect*), 293
is_java_method() (*in module robot.utils.robotinspect*), 293
is_list_like() (*in module robot.utils.robottypes2*), 296
is_list_var() (*in module robot.variables.isvar*), 299
is_list_variable() (*robot.variables.splitter.VariableSplitter method*), 301
is_number() (*in module robot.utils.robottypes2*), 295
is_process_running() (*robot.libraries.Process.Process method*), 72
is_scalar_var() (*in module robot.variables.isvar*), 299
is_set() (*robot.parsing.model.ForLoop method*), 212
is_set() (*robot.parsing.model.Step method*), 212
is_set() (*robot.parsing.model.Variable method*), 211
is_set() (*robot.parsing.settings.Arguments method*), 216
is_set() (*robot.parsing.settings.Documentation method*), 214
is_set() (*robot.parsing.settings.Fixture method*), 215
is_set() (*robot.parsing.settings.Library method*), 217
is_set() (*robot.parsing.settings.Metadata method*), 217
is_set() (*robot.parsing.settings.Resource method*), 217
is_set() (*robot.parsing.settings.Return method*), 216
is_set() (*robot.parsing.settings.Setting method*), 214
is_set() (*robot.parsing.settings.Tags method*), 216
is_set() (*robot.parsing.settings.Template method*), 215
is_set() (*robot.parsing.settings.Timeout method*), 215
is_set() (*robot.parsing.settings.Variables method*), 217
is_started() (*robot.parsing.model.TestCaseTable method*), 210
is_string() (*in module robot.utils.robottypes2*), 296
is_truthy() (*in module robot.utils.robottypes*), 295
is_unicode() (*in module robot.utils.robottypes2*), 296
is_var() (*in module robot.variables.isvar*), 299
is_variable() (*robot.variables.splitter.VariableSplitter method*), 301
isatty() (*in module robot.utils.compat*), 284
IsLogged (*class in robot.output.loggerhelper*), 200
isreadable() (*robot.utils.unic.PrettyRepr method*), 297
isrecursive() (*robot.utils.unic.PrettyRepr method*), 297
ItemList (*class in robot.model.itemlist*), 174
items() (*robot.model.metadata.Metadata method*),

177
items() (*robot.utils.dotdict.DotDict method*), 285
items() (*robot.utils.normalizing.NormalizedDict method*), 292
iteritems() (*robot.model.metadata.Metadata method*), 177
iteritems() (*robot.utils.dotdict.DotDict method*), 285
iteritems() (*robot.utils.normalizing.NormalizedDict method*), 292
iterkeys() (*robot.model.metadata.Metadata method*), 177
iterkeys() (*robot.utils.dotdict.DotDict method*), 285
iterkeys() (*robot.utils.normalizing.NormalizedDict method*), 292
itervalues() (*robot.model.metadata.Metadata method*), 177
itervalues() (*robot.utils.dotdict.DotDict method*), 285
itervalues() (*robot.utils.normalizing.NormalizedDict method*), 292

J

JavaArgumentParser (*class in robot.running.arguments.argumentparser*), 258
JavaCapturer (*class in robot.running.outputcapture*), 273
JavaDocBuilder (*class in robot.libdocpkg.javabuilder*), 22
JavaDocBuilder() (*in module robot.libdocpkg.builder*), 21
JavaErrorDetails (*class in robot.utils.error*), 287
join_command_line() (*robot.libraries.Process.Process method*), 74
join_path() (*robot.libraries.OperatingSystem.OperatingSystem method*), 66
join_paths() (*robot.libraries.OperatingSystem.OperatingSystem method*), 66
join_string_with_value() (*robot.parsing.settings.StringValueJoiner method*), 214
js_result (*robot.reporting.resultwriter.Results attribute*), 224
JsBuildingContext (*class in robot.reporting.jsbuildingcontext*), 220
JsExecutionResult (*class in robot.reporting.jsexecutionresult*), 221
JsFileWriter (*class in robot.htmldata.htmlfilewriter*), 19
JsModelBuilder (*class in robot.reporting.jsmodelbuilders*), 221
JsonConverter (*class in robot.libdocpkg.htmlwriter*), 21
JsonConverter (*class in robot.testdoc*), 316
JsonDumper (*class in robot.htmldata.jsonwriter*), 20
JsonWriter (*class in robot.htmldata.jsonwriter*), 20
JsResultWriter (*class in robot.reporting.jswriter*), 222

K

keep_in_dictionary() (*robot.libraries.Collections.Collections method*), 51
keys() (*robot.libraries.dialogs_py.InputDialog method*), 122
keys() (*robot.libraries.dialogs_py.MessageDialog method*), 109
keys() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 148
keys() (*robot.libraries.dialogs_py.PassFailDialog method*), 161
keys() (*robot.libraries.dialogs_py.SelectionDialog method*), 135
keys() (*robot.model.metadata.Metadata method*), 177
keys() (*robot.utils.dotdict.DotDict method*), 285
keys() (*robot.utils.normalizing.NormalizedDict method*), 292
Keyword (*class in robot.model.keyword*), 174
Keyword (*class in robot.result.model*), 241
Keyword (*class in robot.running.model*), 265
keyword (*robot.parsing.settings.Fixture attribute*), 215
keyword() (*in module robot.api.deco*), 12
keyword_class (*robot.model.keyword.Keyword attribute*), 174
keyword_class (*robot.model.testcase.TestCase attribute*), 186
keyword_class (*robot.model.testsuite.TestSuite attribute*), 187
keyword_class (*robot.result.model.Keyword attribute*), 243
keyword_class (*robot.result.model.TestCase attribute*), 243
keyword_class (*robot.result.model.TestSuite attribute*), 244
keyword_class (*robot.running.model.ForLoop attribute*), 267
keyword_class (*robot.running.model.Keyword attribute*), 266
keyword_class (*robot.running.model.TestCase attribute*), 268
keyword_class (*robot.running.model.TestSuite attribute*), 269
keyword_marker() (*robot.output.console.verbose.VerboseWriter method*), 195

keyword_should_exist() (*robot.libraries.BuiltIn.BuiltIn method*), 32
keyword_timeout (*robot.errors.TimeoutError attribute*), 307
KEYWORD_TYPE (*robot.model.keyword.Keyword attribute*), 174
KEYWORD_TYPE (*robot.result.model.Keyword attribute*), 242
KEYWORD_TYPE (*robot.running.model.ForLoop attribute*), 267
KEYWORD_TYPE (*robot.running.model.Keyword attribute*), 266
KeywordBuilder (*class in robot.reporting.jsmodelbuilders*), 221
KeywordCallTemplate (*class in robot.running.arguments.argumentmapper*), 258
KeywordDoc (*class in robot.libdocpkg.model*), 22
KeywordDocBuilder (*class in robot.libdocpkg.robotbuilder*), 23
KeywordError, 307
KeywordHandler (*class in robot.result.xmlelementhandlers*), 253
KeywordMarker (*class in robot.output.console.verbose*), 195
KeywordMatcher (*class in robot.libdocpkg.consoleviewer*), 21
KeywordRecommendationFinder (*class in robot.running.namespace*), 273
KeywordRemover() (*in module robot.result.keywordremover*), 232
Keywords (*class in robot.model.keyword*), 175
keywords (*robot.libdocpkg.model.LibraryDoc attribute*), 22
keywords (*robot.model.keyword.Keyword attribute*), 175
keywords (*robot.model.testcase.TestCase attribute*), 186
keywords (*robot.model.testsuite.TestSuite attribute*), 188
keywords (*robot.parsing.model.ResourceFile attribute*), 208
keywords (*robot.parsing.model.TestCaseFile attribute*), 208
keywords (*robot.parsing.model.TestDataDirectory attribute*), 209
keywords (*robot.result.model.Keyword attribute*), 243
keywords (*robot.result.model.TestCase attribute*), 244
keywords (*robot.result.model.TestSuite attribute*), 246
keywords (*robot.running.model.ForLoop attribute*), 267
keywords (*robot.running.model.Keyword attribute*), 266
keywords (*robot.running.model.ResourceFile attribute*), 272
keywords (*robot.running.model.TestCase attribute*), 269
keywords (*robot.running.model.TestSuite attribute*), 271
keywords (*robot.running.model.UserKeyword attribute*), 272
KeywordStatusHandler (*class in robot.result.xmlelementhandlers*), 254
KeywordStore (*class in robot.running.namespace*), 273
KeywordTable (*class in robot.parsing.model*), 211
KeywordTablePopulator (*class in robot.parsing.tablepopulators*), 219
KeywordTimeout (*class in robot.running.timeouts*), 260
KILL_TIMEOUT (*robot.libraries.Process.Process attribute*), 72
kwname (*robot.result.model.Keyword attribute*), 241

L

length_should_be() (*robot.libraries.BuiltIn.BuiltIn method*), 32
level (*robot.model.message.Message attribute*), 176
level (*robot.output.loggerhelper.Message attribute*), 200
level (*robot.result.model.Message attribute*), 241
LibDoc (*class in robot.libdoc*), 311
libdoc() (*in module robot.libdoc*), 312
libdoc_cli() (*in module robot.libdoc*), 311
LibdocHtmlWriter (*class in robot.libdocpkg.htmlwriter*), 21
LibdocModelWriter (*class in robot.libdocpkg.htmlwriter*), 21
LibdocOutput (*class in robot.libdocpkg.output*), 22
LibdocWriter() (*in module robot.libdocpkg.writer*), 23
LibdocXmlWriter (*class in robot.libdocpkg.xmlwriter*), 23
libname (*robot.result.model.Keyword attribute*), 241
libname (*robot.running.librarykeywordrunner.EmbeddedArgumentsRunner attribute*), 264
libname (*robot.running.librarykeywordrunner.LibraryKeywordRunner attribute*), 264
libname (*robot.running.librarykeywordrunner.RunKeywordRunner attribute*), 264
libname (*robot.running.userkeywordrunner.EmbeddedArgumentsRunner attribute*), 279
libname (*robot.running.userkeywordrunner.UserKeywordRunner attribute*), 279
libraries (*robot.running.namespace.Namespace attribute*), 272

Library (*class in robot.parsing.settings*), 217
library (*robot.running.handlers.EmbeddedArgumentsHandler attribute*), 263
library (*robot.running.librarykeywordrunner.EmbeddedArgumentsRunner attribute*), 264
library (*robot.running.librarykeywordrunner.LibraryKeywordRunner attribute*), 264
library (*robot.running.librarykeywordrunner.RunKeywordRunner attribute*), 264
library() (*robot.model.imports.Imports method*), 173
LibraryDoc (*class in robot.libdocpkg.model*), 22
LibraryDocBuilder (*class in robot.libdocpkg.robotbuilder*), 22
LibraryDocumentation() (*in module robot.libdocpkg*), 21
LibraryKeywordRunner (*class in robot.running.librarykeywordrunner*), 264
LibraryListenerMethods (*class in robot.output.listenermethods*), 197
LibraryListeners (*class in robot.output.listeners*), 198
LibraryScope() (*in module robot.running.libraryscopes*), 264
lift() (*robot.libraries.dialogs_py.InputDialog method*), 123
lift() (*robot.libraries.dialogs_py.MessageDialog method*), 109
lift() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 149
lift() (*robot.libraries.dialogs_py.PassFailDialog method*), 162
lift() (*robot.libraries.dialogs_py.SelectionDialog method*), 136
line_sep() (*robot.tidy.ArgumentValidator method*), 318
LineFormatter (*class in robot.utils.htmlformatters*), 287
LineWriter (*class in robot.htmldata.htmlfilewriter*), 19
link() (*robot.reporting.jsbuildingcontext.JsBuildingContext method*), 220
LinkFormatter (*class in robot.utils.htmlformatters*), 287
links (*robot.model.stats.TagStat attribute*), 182
list() (*robot.libdocpkg.consoleviewer.ConsoleViewer method*), 21
list_directories_in_directory() (*robot.libraries.OperatingSystem.OperatingSystem method*), 68
list_directory() (*robot.libraries.OperatingSystem.OperatingSystem method*), 68
list_files_in_directory() (*robot.libraries.OperatingSystem.OperatingSystem method*), 68
list_should_contain_sub_list() (*robot.libraries.Collections.Collections method*), 51
list_should_contain_value() (*robot.libraries.Collections.Collections method*), 51
list_should_not_contain_duplicates() (*robot.libraries.Collections.Collections method*), 51
list_should_not_contain_value() (*robot.libraries.Collections.Collections method*), 52
listener() (*robot.libraries.Telnet.TelnetConnection method*), 91
ListenerArguments (*class in robot.output.listenerarguments*), 196
ListenerMethod (*class in robot.output.listenermethods*), 197
ListenerMethods (*class in robot.output.listenermethods*), 197
ListenerProxy (*class in robot.output.listeners*), 198
Listeners (*class in robot.output.listeners*), 198
listeners (*robot.conf.settings.RobotSettings attribute*), 17
ListFormatter (*class in robot.utils.htmlformatters*), 289
lists_should_be_equal() (*robot.libraries.Collections.Collections method*), 52
ListVariableTableValue (*class in robot.variables.tablesetter*), 302
Location (*class in robot.libraries.XML*), 104
log (*robot.conf.settings.RebotSettings attribute*), 18
log (*robot.conf.settings.RobotSettings attribute*), 18
log() (*robot.libraries.BuiltIn.BuiltIn method*), 32
log_config (*robot.conf.settings.RebotSettings attribute*), 18
log_dictionary() (*robot.libraries.Collections.Collections method*), 52
log_element() (*robot.libraries.XML.XML method*), 103
log_environment_variables() (*robot.libraries.OperatingSystem.OperatingSystem method*), 66
log_file() (*robot.libraries.OperatingSystem.OperatingSystem method*), 62
log_level (*robot.conf.settings.RebotSettings attribute*), 18
log_level (*robot.conf.settings.RobotSettings attribute*), 18
log_list() (*robot.libraries.Collections.Collections method*), 52
log_many() (*robot.libraries.BuiltIn.BuiltIn method*), 33

log_message() (*robot.output.listeners.LibraryListeners method*), 198
log_message() (*robot.output.listeners.Listeners method*), 198
log_message() (*robot.output.logger.Logger method*), 199
log_message() (*robot.output.xmllogger.XmlLogger method*), 203
log_message() (*robot.reporting.outputwriter.OutputWriter method*), 223
log_output() (*robot.output.logger.Logger method*), 199
log_to_console() (*robot.libraries.BuiltIn.BuiltIn method*), 33
log_variables() (*robot.libraries.BuiltIn.BuiltIn method*), 33
Logger (*class in robot.output.logger*), 198
LoggerProxy (*class in robot.output.logger*), 199
login() (*robot.libraries.Telnet.TelnetConnection method*), 89
LogWriter (*class in robot.reporting.logreportwriters*), 222
longname (*robot.model.testcase.TestCase attribute*), 186
longname (*robot.model.testsuite.TestSuite attribute*), 187
longname (*robot.result.model.TestCase attribute*), 244
longname (*robot.result.model.TestSuite attribute*), 247
longname (*robot.running.librarykeywordrunner.EmbeddedArgumentsRunner attribute*), 264
longname (*robot.running.librarykeywordrunner.LibraryKeywordRunner attribute*), 264
longname (*robot.running.librarykeywordrunner.RunKeywordRunner attribute*), 264
longname (*robot.running.model.TestCase attribute*), 269
longname (*robot.running.model.TestSuite attribute*), 271
longname (*robot.running.usererrorhandler.UserErrorHandler attribute*), 278
longname (*robot.running.userkeyword.EmbeddedArgumentsHandler attribute*), 279
longname (*robot.running.userkeyword.UserKeywordHandler attribute*), 278
longname (*robot.running.userkeywordrunner.EmbeddedArgumentsRunner attribute*), 279
longname (*robot.running.userkeywordrunner.UserKeywordRunner attribute*), 279
lower() (*in module robot.utils.normalizing*), 292
lower() (*robot.libraries.dialogs_py.InputDialog method*), 123
lower() (*robot.libraries.dialogs_py.MessageDialog method*), 110
lower() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 149
lower() (*robot.libraries.dialogs_py.PassFailDialog method*), 162
lower() (*robot.libraries.dialogs_py.SelectionDialog method*), 136

M

main() (*robot.libdoc.LibDoc method*), 311
main() (*robot.rebot.Rebot method*), 313
main() (*robot.run.RobotFramework method*), 314
main() (*robot.testdoc.TestDoc method*), 316
main() (*robot.tidy.TidyCommandLine method*), 318
main() (*robot.utils.application.Application method*), 280
mainloop() (*robot.libraries.dialogs_py.InputDialog method*), 123
mainloop() (*robot.libraries.dialogs_py.MessageDialog method*), 110
mainloop() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 149
mainloop() (*robot.libraries.dialogs_py.PassFailDialog method*), 162
mainloop() (*robot.libraries.dialogs_py.SelectionDialog method*), 136
make_connection() (*robot.libraries.Remote.TimeoutHTTPSTransport method*), 76
make_connection() (*robot.libraries.Remote.TimeoutHTTPTransport method*), 76
map() (*robot.running.arguments.argumentmapper.ArgumentMapper method*), 258
map() (*robot.running.arguments.argumentspec.ArgumentSpec method*), 259
MappingDumper (*class in robot.htmldata.jsonwriter*), 20
mark() (*robot.output.console.verbose.KeywordMarker method*), 195
match() (*robot.model.namepatterns.SuiteNamePatterns method*), 179
match() (*robot.model.namepatterns.TestNamePatterns method*), 179
match() (*robot.model.stats.CombinedTagStat method*), 182
match() (*robot.model.stats.CriticalTagStat method*), 182
match() (*robot.model.tags.AndTagPattern method*), 184
match() (*robot.model.tags.NotTagPattern method*), 184
match() (*robot.model.tags.OrTagPattern method*), 184
match() (*robot.model.tags.SingleTagPattern method*), 183
match() (*robot.model.tags.TagPatterns method*), 183
match() (*robot.model.tags.Tags method*), 183

match() (*robot.model.tagstatistics.TagStatDoc method*), 185
match() (*robot.model.tagstatistics.TagStatLink method*), 185
match() (*robot.parsing.model.OldStyleSettingAndVariableTableHeaderMatcher method*), 212
match() (*robot.parsing.model.OldStyleTestAndKeywordTableHeaderMatcher method*), 212
match() (*robot.result.flattenkeywordmatcher.FlattenByNameMatcher method*), 232
match() (*robot.result.flattenkeywordmatcher.FlattenByTagMatcher method*), 232
match() (*robot.result.flattenkeywordmatcher.FlattenByTypeMatcher method*), 232
match() (*robot.utils.htmlformatters.HeaderFormatter method*), 288
match() (*robot.utils.htmlformatters.RulerFormatter method*), 288
match() (*robot.utils.match.Matcher method*), 291
match() (*robot.utils.match.MultiMatcher method*), 291
match_any() (*robot.utils.match.Matcher method*), 291
match_any() (*robot.utils.match.MultiMatcher method*), 291
Matcher (*class in robot.utils.match*), 291
matches() (*robot.running.handlers.EmbeddedArgumentsHandler method*), 263
matches() (*robot.running.userkeyword.EmbeddedArgumentsHandler method*), 279
max_error_lines (*robot.conf.settings.RobotSettings attribute*), 17
maxargs (*robot.running.arguments.argumentspec.ArgumentSpec attribute*), 259
maxsize() (*robot.libraries.dialogs_py.InputDialog method*), 123
maxsize() (*robot.libraries.dialogs_py.MessageDialog method*), 110
maxsize() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 149
maxsize() (*robot.libraries.dialogs_py.PassFailDialog method*), 162
maxsize() (*robot.libraries.dialogs_py.SelectionDialog method*), 136
merge (*robot.conf.settings.RebotSettings attribute*), 18
merge() (*robot.result.merger.Merger method*), 238
Merger (*class in robot.result.merger*), 238
Message (*class in robot.model.message*), 176
Message (*class in robot.output.loggerhelper*), 200
Message (*class in robot.result.model*), 241
message (*robot.errors.ContinueForLoop attribute*), 310
message (*robot.errors.DataError attribute*), 307
message (*robot.errors.ExecutionFailed attribute*), 308
message (*robot.errors.ExecutionFailures attribute*), 308
message (*robot.errors.ExecutionPassed attribute*), 309
message (*robot.errors.ExecutionStatus attribute*), 308
message (*robot.errors.ExitForLoop attribute*), 310
message (*robot.errors.FrameworkError attribute*), 306
message (*robot.errors.HandlerExecutionFailed attribute*), 308
message (*robot.errors.Information attribute*), 307
message (*robot.errors.KeywordError attribute*), 307
message (*robot.errors.PassExecution attribute*), 309
message (*robot.errors.RemoteError attribute*), 311
message (*robot.errors.ReturnFromKeyword attribute*), 310
message (*robot.errors.RobotError attribute*), 306
message (*robot.errors.TimeoutError attribute*), 307
message (*robot.errors.UserKeywordExecutionFailed attribute*), 309
message (*robot.errors.VariableError attribute*), 307
message (*robot.libraries.BuiltIn.RobotNotRunningError attribute*), 46
message (*robot.libraries.Telnet.NoMatchError attribute*), 93
message (*robot.model.message.Message attribute*), 176
message (*robot.model.totalstatistics.TotalStatistics attribute*), 189
message (*robot.output.loggerhelper.Message attribute*), 200
message (*robot.parsing.populators.NoTestsFound attribute*), 213
message (*robot.result.model.Keyword attribute*), 242
message (*robot.result.model.Message attribute*), 241
message (*robot.result.model.TestCase attribute*), 243
message (*robot.result.model.TestSuite attribute*), 244
message (*robot.running.status.ParentMessage attribute*), 277
message (*robot.running.status.SuiteMessage attribute*), 277
message (*robot.running.status.SuiteStatus attribute*), 276
message (*robot.running.status.TestMessage attribute*), 277
message (*robot.running.status.TestStatus attribute*), 276
message (*robot.utils.error.JavaErrorDetails attribute*), 287
message (*robot.utils.error.PythonErrorDetails attribute*), 287
message() (*robot.output.console.dotted.DottedOutput method*), 192
message() (*robot.output.console.quiet.QuietOutput method*), 194
message() (*robot.output.console.verbose.VerboseOutput method*), 195
message() (*robot.output.console.verbose.VerboseWriter method*), 195
message() (*robot.output.filelogger.FileLogger method*), 195

message() (*robot.output.logger.Logger method*), 199
message() (*robot.output.loggerhelper.AbstractLogger method*), 200
message() (*robot.output.output.Output method*), 201
message() (*robot.output.xmllogger.XmlLogger method*), 203
message() (*robot.reporting.outputwriter.OutputWriter method*), 223
message_class (*robot.model.keyword.Keyword attribute*), 174
message_class (*robot.result.executionerrors.ExecutionErrors attribute*), 229
message_class (*robot.result.model.Keyword attribute*), 241
message_class (*robot.running.model.ForLoop attribute*), 268
message_class (*robot.running.model.Keyword attribute*), 265
message_level() (*robot.reporting.jsbuildingcontext.JsBuildingContext method*), 220
MessageArguments (*class in robot.output.listenerarguments*), 196
MessageBuilder (*class in robot.reporting.jsmodelbuilders*), 221
MessageDialog (*class in robot.libraries.dialogs_py*), 104
MessageFilter (*class in robot.result.messagefilter*), 239
MessageHandler (*class in robot.result.xmlelementhandlers*), 254
Messages (*class in robot.model.message*), 177
messages (*robot.model.keyword.Keyword attribute*), 175
messages (*robot.result.executionerrors.ExecutionErrors attribute*), 229
messages (*robot.result.model.Keyword attribute*), 243
messages (*robot.running.model.ForLoop attribute*), 268
messages (*robot.running.model.Keyword attribute*), 266
meta_start() (*robot.parsing.htmlreader.HtmlReader method*), 206
Metadata (*class in robot.model.metadata*), 177
Metadata (*class in robot.parsing.settings*), 216
metadata (*robot.model.testsuite.TestSuite attribute*), 187
metadata (*robot.result.model.TestSuite attribute*), 247
metadata (*robot.running.model.TestSuite attribute*), 271
MetadataHandler (*class in robot.result.xmlelementhandlers*), 254
MetadataItemHandler (*class in robot.result.xmlelementhandlers*), 255
MetadataList (*class in robot.parsing.settings*), 218
MetadataPopulator (*class in robot.parsing.tablepopulators*), 219
minargs (*robot.running.arguments.argumentspec.ArgumentSpec attribute*), 259
minsize() (*robot.libraries.dialogs_py.InputDialog method*), 123
minsize() (*robot.libraries.dialogs_py.MessageDialog method*), 110
minsize() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 149
minsize() (*robot.libraries.dialogs_py.PassFailDialog method*), 162
minsize() (*robot.libraries.dialogs_py.SelectionDialog method*), 136
mode_and_arguments() (*robot.tidy.ArgumentValidator method*), 318
ModelCombiner (*class in robot.running.runner*), 275
ModelModifier (*class in robot.model.modifier*), 178
ModelObject (*class in robot.model.modelobject*), 178
ModelWriter (*class in robot.htmldata.htmlfilewriter*), 19
move_directory() (*robot.libraries.OperatingSystem.OperatingSystem method*), 65
move_file() (*robot.libraries.OperatingSystem.OperatingSystem method*), 65
move_files() (*robot.libraries.OperatingSystem.OperatingSystem method*), 65
mro() (*robot.utils.setter.SetterAwareType method*), 296
msg() (*robot.libraries.Telnet.TelnetConnection method*), 91
mt_interact() (*robot.libraries.Telnet.TelnetConnection method*), 92
MultiMatcher (*class in robot.utils.match*), 291
MultipleSelectionDialog (*class in robot.libraries.dialogs_py*), 143

N

name (*robot.model.keyword.Keyword attribute*), 175
name (*robot.model.stats.Stat attribute*), 181
name (*robot.model.testcase.TestCase attribute*), 186
name (*robot.model.testsuite.TestSuite attribute*), 187
name (*robot.output.pyloggingconf.RobotHandler attribute*), 202
name (*robot.parsing.model.InitFileSettingTable attribute*), 210
name (*robot.parsing.model.KeywordTable attribute*), 211
name (*robot.parsing.model.ResourceFile attribute*), 208
name (*robot.parsing.model.ResourceFileSettingTable attribute*), 209
name (*robot.parsing.model.TestCaseFile attribute*), 208
name (*robot.parsing.model.TestCaseFileSettingTable attribute*), 209
name (*robot.parsing.model.TestCaseTable attribute*), 210

name (*robot.parsing.model.TestDataDirectory attribute*), 209
name (*robot.parsing.model.VariableTable attribute*), 210
name (*robot.result.model.Keyword attribute*), 242
name (*robot.result.model.TestCase attribute*), 244
name (*robot.result.model.TestSuite attribute*), 247
name (*robot.running.dynamicmethods.GetKeywordArguments attribute*), 263
name (*robot.running.dynamicmethods.GetKeywordDocumentation attribute*), 262
name (*robot.running.dynamicmethods.GetKeywordNames attribute*), 262
name (*robot.running.dynamicmethods.GetKeywordTags attribute*), 263
name (*robot.running.dynamicmethods.GetKeywordTypes attribute*), 263
name (*robot.running.dynamicmethods.RunKeyword attribute*), 262
name (*robot.running.model.ForLoop attribute*), 268
name (*robot.running.model.Keyword attribute*), 266
name (*robot.running.model.TestCase attribute*), 269
name (*robot.running.model.TestSuite attribute*), 271
NameCell (*class in robot.writer.htmlformatter*), 306
NamedArgumentResolver (*class in robot.running.arguments.argumentresolver*), 259
Namespace (*class in robot.running.namespace*), 272
namespaces (*robot.running.context.ExecutionContexts attribute*), 262
NameSpaceStripper (*class in robot.libraries.XML*), 104
nametowidget() (*robot.libraries.dialogs_py.InputDialog method*), 123
nametowidget() (*robot.libraries.dialogs_py.MessageDialog method*), 110
nametowidget() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 149
nametowidget() (*robot.libraries.dialogs_py.PassFailDialog method*), 162
nametowidget() (*robot.libraries.dialogs_py.SelectionDialog method*), 136
NEW_ENVIRON_IS (*robot.libraries.Telnet.TelnetConnection attribute*), 88
NEW_ENVIRON_VALUE (*robot.libraries.Telnet.TelnetConnection attribute*), 88
NEW_ENVIRON_VAR (*robot.libraries.Telnet.TelnetConnection attribute*), 88
new_suite_scope() (*robot.output.listenermethods.LibraryListenerMethods method*), 197
new_suite_scope() (*robot.output.listeners.LibraryListeners method*), 198
newline (*robot.utils.htmlformatters.LineFormatter attribute*), 288
no_dynamic_method() (*in module robot.running.dynamicmethods*), 262
no_operation() (*robot.libraries.BuiltIn.BuiltIn method*), 33
NoConnection (*class in robot.utils.connectioncache*), 285
NoHighlighting (*class in robot.output.console.highlighting*), 194
NoMatchError, 93
non_ascii (*robot.libraries.Remote.ArgumentCoercer attribute*), 75
non_critical (*robot.model.stats.TagStat attribute*), 182
non_critical (*robot.model.tagstatistics.TagStatistics attribute*), 185
non_critical_tags (*robot.conf.settings.RebotSettings attribute*), 18
non_critical_tags (*robot.conf.settings.RobotSettings attribute*), 18
NonDottedImporter (*class in robot.utils.importer*), 289
none_shall_pass() (*in module robot.libraries.Easter*), 59
NoneDumper (*class in robot.htmldata.jsonwriter*), 20
NoOutput (*class in robot.output.console.quiet*), 194
NoReturnValueResolver (*class in robot.variables.assigner*), 297
normal (*robot.model.keyword.Keywords attribute*), 176
normalize() (*in module robot.utils.normalizing*), 292
normalize_path() (*robot.libraries.OperatingSystem.OperatingSystem method*), 67
normalized_flavors (*robot.parsing.model.ForLoop attribute*), 212
NormalizedDict (*class in robot.utils.normalizing*), 292
normpath() (*in module robot.utils.robotpath*), 293
NoTestsFound, 213
NotSet (*class in robot.libraries.Collections*), 47
NotTagPattern (*class in robot.model.tags*), 184
NullAligner (*class in robot.writer.aligners*), 303
NullMarkupWriter (*class in robot.utils.markupwriters*), 291
NullNamedArgumentResolver (*class in robot.running.arguments.argumentresolver*), 259
NullPopulator (*class in robot.parsing.tablepopulators*), 218
NumberFinder (*class in robot.variables.finders*), 298
numerator (*robot.reporting.stringcache.StringIndex attribute*), 225

O

OldStyleSettingAndVariableTableHeaderMatcher (*class in robot.parsing.model*), 212
OldStyleTestAndKeywordTableHeaderMatcher (*class in robot.parsing.model*), 212
OneReturnValueResolver (*class in robot.variables.assigner*), 298
open() (*robot.libraries.Telnet.TelnetConnection method*), 92
open_connection() (*robot.libraries.Telnet.Telnet method*), 87
OperatingSystem (*class in robot.libraries.OperatingSystem*), 59
option_add() (*robot.libraries.dialogs_py.InputDialog method*), 123
option_add() (*robot.libraries.dialogs_py.MessageDialog method*), 110
option_add() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 149
option_add() (*robot.libraries.dialogs_py.PassFailDialog method*), 162
option_add() (*robot.libraries.dialogs_py.SelectionDialog method*), 136
option_clear() (*robot.libraries.dialogs_py.InputDialog method*), 123
option_clear() (*robot.libraries.dialogs_py.MessageDialog method*), 110
option_clear() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 149
option_clear() (*robot.libraries.dialogs_py.PassFailDialog method*), 162
option_clear() (*robot.libraries.dialogs_py.SelectionDialog method*), 136
option_get() (*robot.libraries.dialogs_py.InputDialog method*), 123
option_get() (*robot.libraries.dialogs_py.MessageDialog method*), 110
option_get() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 149
option_get() (*robot.libraries.dialogs_py.PassFailDialog method*), 162
option_get() (*robot.libraries.dialogs_py.SelectionDialog method*), 136
option_readfile() (*robot.libraries.dialogs_py.InputDialog method*), 123
option_readfile() (*robot.libraries.dialogs_py.MessageDialog method*), 110
option_readfile() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 149
option_readfile() (*robot.libraries.dialogs_py.PassFailDialog method*), 162
option_readfile() (*robot.libraries.dialogs_py.SelectionDialog method*), 136
option_spec (*robot.parsing.restsupport.CaptureRobotData attribute*), 214
optional_arguments (*robot.parsing.restsupport.CaptureRobotData attribute*), 214
OrTagPattern (*class in robot.model.tags*), 184
Output (*class in robot.output.output*), 201
output (*robot.conf.settings.RebotSettings attribute*), 18
output (*robot.conf.settings.RobotSettings attribute*), 18
output() (*robot.output.console.verbose.VerboseWriter method*), 195
output_directory (*robot.conf.settings.RebotSettings attribute*), 18
output_directory (*robot.conf.settings.RobotSettings attribute*), 18
output_file() (*robot.output.console.dotted.DottedOutput method*), 193
output_file() (*robot.output.console.verbose.VerboseOutput method*), 195
output_file() (*robot.output.filelogger.FileLogger method*), 196
output_file() (*robot.output.listeners.LibraryListeners method*), 198
output_file() (*robot.output.listeners.Listeners method*), 198
output_file() (*robot.output.logger.Logger method*), 199
OutputCapturer (*class in robot.running.outputcapture*), 273
OutputWriter (*class in robot.reporting.outputwriter*), 222
overrideredirect() (*robot.libraries.dialogs_py.InputDialog method*), 123
overrideredirect() (*robot.libraries.dialogs_py.MessageDialog method*), 110
overrideredirect() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 149
overrideredirect() (*robot.libraries.dialogs_py.PassFailDialog method*), 162
overrideredirect() (*robot.libraries.dialogs_py.SelectionDialog method*), 136

P

pack_propagate() (*robot.libraries.dialogs_py.InputDialog method*), 123

pack_propagate() (*robot.libraries.dialogs_py.MessageDialog method*), 110
pack_propagate() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 149
pack_propagate() (*robot.libraries.dialogs_py.PassFailDialog method*), 162
pack_propagate() (*robot.libraries.dialogs_py.SelectionDialog method*), 136
pack_slaves() (*robot.libraries.dialogs_py.InputDialog method*), 123
pack_slaves() (*robot.libraries.dialogs_py.MessageDialog method*), 110
pack_slaves() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 149
pack_slaves() (*robot.libraries.dialogs_py.PassFailDialog method*), 162
pack_slaves() (*robot.libraries.dialogs_py.SelectionDialog method*), 136
pad_console_length() (*in module robot.utils.text*), 296
ParagraphFormatter (*class in robot.utils.htmlformatters*), 288
parent (*robot.model.keyword.Keyword attribute*), 175
parent (*robot.model.message.Message attribute*), 176
parent (*robot.model.testcase.TestCase attribute*), 186
parent (*robot.model.testsuite.TestSuite attribute*), 187
parent (*robot.output.loggerhelper.Message attribute*), 200
parent (*robot.result.model.Keyword attribute*), 243
parent (*robot.result.model.Message attribute*), 241
parent (*robot.result.model.TestCase attribute*), 244
parent (*robot.result.model.TestSuite attribute*), 247
parent (*robot.running.model.ForLoop attribute*), 268
parent (*robot.running.model.Keyword attribute*), 266
parent (*robot.running.model.TestCase attribute*), 269
parent (*robot.running.model.TestSuite attribute*), 271
ParentMessage (*class in robot.running.status*), 277
parse() (*robot.running.arguments.argumentparser.DynamicArgumentParser method*), 258
parse() (*robot.running.arguments.argumentparser.JavaArgumentParser method*), 258
parse() (*robot.running.arguments.argumentparser.PythonArgumentParser method*), 258
parse() (*robot.running.arguments.argumentparser.UserKeywordArgumentParser method*), 258
parse() (*robot.running.arguments.embedded.EmbeddedArgumentParser method*), 260
parse_args() (*robot.utils.argumentparser.ArgumentParser method*), 280
parse_arguments() (*robot.libdoc.LibDoc method*), 311
parse_arguments() (*robot.rebot.Rebot method*), 313
parse_arguments() (*robot.run.RobotFramework method*), 314
parse_arguments() (*robot.testdoc.TestDoc method*), 316
parse_arguments() (*robot.tidy.TidyCommandLine method*), 318
parse_arguments() (*robot.utils.application.Application method*), 280
parse_bogus_comment() (*robot.parsing.htmlreader.HtmlReader method*), 207
parse_comment() (*robot.parsing.htmlreader.HtmlReader method*), 207
parse_declaration() (*robot.parsing.htmlreader.HtmlReader method*), 207
parse_endtag() (*robot.parsing.htmlreader.HtmlReader method*), 207
parse_html_declaration() (*robot.parsing.htmlreader.HtmlReader method*), 207
parse_marked_section() (*robot.parsing.htmlreader.HtmlReader method*), 207
parse_pi() (*robot.parsing.htmlreader.HtmlReader method*), 207
parse_response() (*robot.libraries.Remote.TimeoutHTTPSTransport method*), 76
parse_response() (*robot.libraries.Remote.TimeoutHTTPTransport method*), 76
parse_starttag() (*robot.parsing.htmlreader.HtmlReader method*), 207
parse_time() (*in module robot.utils.robottime*), 295
parse_xml() (*robot.libraries.XML.XML method*), 97
pass_execution() (*robot.libraries.BuiltIn.BuiltIn method*), 33
pass_execution_if() (*robot.libraries.BuiltIn.BuiltIn method*), 34
passed (*robot.model.stats.Stat attribute*), 181
passed (*robot.result.model.Keyword attribute*), 242
passed (*robot.result.model.TestCase attribute*), 243
passed (*robot.result.model.TestSuite attribute*), 245
PassExecutionRemover (*class in robot.result.keywordremover*), 233
PassExecution, 309
PassFailDialog (*class in robot.libraries.dialogs_py*), 156
path_to_url() (*in module robot.utils.robotpath*), 293
pause_execution() (*in module robot.libraries.Dialogs*), 58
pformat() (*robot.utils.unic.PrettyRepr method*), 297
PipeFormatter (*class in robot.writer.formatters*), 305
PipeSeparatedTxtWriter (*class in*

robot.writer.filewriters), 305
place_slaves() (*robot.libraries.dialogs_py.InputDialog method*), 123
place_slaves() (*robot.libraries.dialogs_py.MessageDialog method*), 110
place_slaves() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 149
place_slaves() (*robot.libraries.dialogs_py.PassFailDialog method*), 162
place_slaves() (*robot.libraries.dialogs_py.SelectionDialog method*), 136
plural_or_not() (*in module robot.utils.misc*), 291
pop() (*robot.model.imports.Imports method*), 173
pop() (*robot.model.itemlist.ItemList method*), 174
pop() (*robot.model.keyword.Keywords method*), 176
pop() (*robot.model.message.Messages method*), 177
pop() (*robot.model.metadata.Metadata method*), 177
pop() (*robot.model.testcase.TestCases method*), 187
pop() (*robot.model.testsuite.TestSuites method*), 189
pop() (*robot.utils.dotdict.DotDict method*), 285
pop() (*robot.utils.normalizing.NormalizedDict method*), 292
pop_from_dictionary() (*robot.libraries.Collections.Collections method*), 52
popen_config (*robot.libraries.Process.ProcessConfiguration attribute*), 75
popitem() (*robot.model.metadata.Metadata method*), 177
popitem() (*robot.utils.dotdict.DotDict method*), 285
popitem() (*robot.utils.normalizing.NormalizedDict method*), 292
populate() (*robot.parsing.model.ResourceFile method*), 208
populate() (*robot.parsing.model.TestCaseFile method*), 208
populate() (*robot.parsing.model.TestDataDirectory method*), 209
populate() (*robot.parsing.populators.FromDirectoryPopulator method*), 213
populate() (*robot.parsing.populators.FromFilePopulator method*), 213
populate() (*robot.parsing.settings.Arguments method*), 216
populate() (*robot.parsing.settings.Documentation method*), 214
populate() (*robot.parsing.settings.Fixture method*), 215
populate() (*robot.parsing.settings.Library method*), 217
populate() (*robot.parsing.settings.Metadata method*), 217
populate() (*robot.parsing.settings.MetadataList method*), 218
populate() (*robot.parsing.settings.Resource method*), 217
populate() (*robot.parsing.settings.Return method*), 216
populate() (*robot.parsing.settings.Setting method*), 214
populate() (*robot.parsing.settings.Tags method*), 216
populate() (*robot.parsing.settings.Template method*), 215
populate() (*robot.parsing.settings.Timeout method*), 215
populate() (*robot.parsing.settings.Variables method*), 218
populate() (*robot.parsing.tablepopulators.DocumentationPopulator method*), 219
populate() (*robot.parsing.tablepopulators.ForLoopPopulator method*), 219
populate() (*robot.parsing.tablepopulators.KeywordTablePopulator method*), 219
populate() (*robot.parsing.tablepopulators.MetadataPopulator method*), 219
populate() (*robot.parsing.tablepopulators.NullPopulator method*), 218
populate() (*robot.parsing.tablepopulators.Populator method*), 218
populate() (*robot.parsing.tablepopulators.SettingPopulator method*), 219
populate() (*robot.parsing.tablepopulators.SettingTablePopulator method*), 218
populate() (*robot.parsing.tablepopulators.StepPopulator method*), 219
populate() (*robot.parsing.tablepopulators.TestCasePopulator method*), 219
populate() (*robot.parsing.tablepopulators.TestTablePopulator method*), 219
populate() (*robot.parsing.tablepopulators.UserKeywordPopulator method*), 219
populate() (*robot.parsing.tablepopulators.VariablePopulator method*), 219
populate() (*robot.parsing.tablepopulators.VariableTablePopulator method*), 218
populate_library() (*robot.parsing.settings.ImportList method*), 218
populate_resource() (*robot.parsing.settings.ImportList method*), 218
populate_variables() (*robot.parsing.settings.ImportList method*), 218
Populator (*class in robot.parsing.tablepopulators*), 218
positionfrom() (*robot.libraries.dialogs_py.InputDialog method*), 123

positionfrom() (*robot.libraries.dialogs_py.MessageDialog method*), 110
positionfrom() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 149
positionfrom() (*robot.libraries.dialogs_py.PassFailDialog method*), 162
positionfrom() (*robot.libraries.dialogs_py.SelectionDialog method*), 136
pprint() (*robot.utils.unic.PrettyRepr method*), 297
pre_rebot_modifiers (*robot.conf.settings.RebotSettings attribute*), 18
pre_rebot_modifiers (*robot.conf.settings.RobotSettings attribute*), 18
pre_run_modifiers (*robot.conf.settings.RobotSettings attribute*), 17
PreformattedFormatter (*class in robot.utils.htmlformatters*), 289
prepr() (*in module robot.utils.unic*), 296
PrettyRepr (*class in robot.utils.unic*), 296
printable_name() (*in module robot.utils.misc*), 291
Process (*class in robot.libraries.Process*), 69
PROCESS (*robot.parsing.htmlreader.HtmlReader attribute*), 206
process() (*robot.utils.argumentparser.ArgFileParser method*), 281
process_empty_suite (*robot.conf.settings.RebotSettings attribute*), 19
process_rawq() (*robot.libraries.Telnet.TelnetConnection method*), 92
process_should_be_running() (*robot.libraries.Process.Process method*), 72
process_should_be_stopped() (*robot.libraries.Process.Process method*), 72
ProcessConfiguration (*class in robot.libraries.Process*), 75
propagate() (*robot.libraries.dialogs_py.InputDialog method*), 123
propagate() (*robot.libraries.dialogs_py.MessageDialog method*), 110
propagate() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 149
propagate() (*robot.libraries.dialogs_py.PassFailDialog method*), 162
propagate() (*robot.libraries.dialogs_py.SelectionDialog method*), 136
protocol() (*robot.libraries.dialogs_py.InputDialog method*), 124
protocol() (*robot.libraries.dialogs_py.MessageDialog method*), 111
protocol() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 150
protocol() (*robot.libraries.dialogs_py.PassFailDialog method*), 163
protocol() (*robot.libraries.dialogs_py.SelectionDialog method*), 137
prune_input() (*robot.reporting.jsbuildingcontext.JsBuildingContext method*), 220
py2to3() (*in module robot.utils.compat*), 284
PythonArgumentParser (*class in robot.running.arguments.argumentparser*), 258
PythonCapturer (*class in robot.running.outputcapture*), 273
PythonErrorDetails (*class in robot.utils.error*), 287
PythonImporter (*class in robot.variables.filesetter*), 298

Q

QuietOutput (*class in robot.output.console.quiet*), 194
quit() (*robot.libraries.dialogs_py.InputDialog method*), 124
quit() (*robot.libraries.dialogs_py.MessageDialog method*), 111
quit() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 150
quit() (*robot.libraries.dialogs_py.PassFailDialog method*), 163
quit() (*robot.libraries.dialogs_py.SelectionDialog method*), 137

R

raise_error() (*robot.utils.connectioncache.NoConnection method*), 285
randomize() (*robot.running.model.TestSuite method*), 270
randomize_seed (*robot.conf.settings.RobotSettings attribute*), 17
randomize_suites (*robot.conf.settings.RobotSettings attribute*), 17
randomize_tests (*robot.conf.settings.RobotSettings attribute*), 17
Randomizer (*class in robot.running.randomizer*), 273
rawq_getchar() (*robot.libraries.Telnet.TelnetConnection method*), 92
read() (*robot.libraries.Telnet.TelnetConnection method*), 90
read() (*robot.libraries.Telnet.TerminalEmulator method*), 93
read() (*robot.parsing.htmlreader.HtmlReader method*), 206
read() (*robot.parsing.tsvreader.TsvReader method*), 220
read() (*robot.utils.utf8reader.Utf8Reader method*), 297

read() (*robot.variables.tablesetter.VariableTableReader method*), 302
read_all() (*robot.libraries.Telnet.TelnetConnection method*), 92
read_eager() (*robot.libraries.Telnet.TelnetConnection method*), 92
read_lazy() (*robot.libraries.Telnet.TelnetConnection method*), 92
read_sb_data() (*robot.libraries.Telnet.TelnetConnection method*), 92
read_some() (*robot.libraries.Telnet.TelnetConnection method*), 92
read_until() (*robot.libraries.Telnet.TelnetConnection method*), 90
read_until() (*robot.libraries.Telnet.TerminalEmulator method*), 93
read_until_prompt() (*robot.libraries.Telnet.TelnetConnection method*), 91
read_until_regexp() (*robot.libraries.Telnet.TelnetConnection method*), 90
read_until_regexp() (*robot.libraries.Telnet.TerminalEmulator method*), 93
read_very_eager() (*robot.libraries.Telnet.TelnetConnection method*), 92
read_very_lazy() (*robot.libraries.Telnet.TelnetConnection method*), 92
readlines() (*robot.utils.utf8reader.Utf8Reader method*), 297
real (*robot.reporting.stringcache.StringIndex attribute*), 225
Rebot (*class in robot.rebot*), 312
rebot() (*in module robot*), 11
rebot() (*in module robot.rebot*), 313
rebot_cli() (*in module robot*), 11
rebot_cli() (*in module robot.rebot*), 313
RebotSettings (*class in robot.conf.settings*), 18
recommend_similar_keywords() (*robot.running.namespace.KeywordRecommendationFinder method*), 273
RecommendationFinder (*class in robot.utils.recommendations*), 293
red() (*robot.output.console.highlighting.AnsiHighlighter method*), 194
red() (*robot.output.console.highlighting.DosHighlighter method*), 194
red() (*robot.output.console.highlighting.NoHighlighting method*), 194
regexp_escape() (*robot.libraries.BuiltIn.BuiltIn method*), 34
register() (*robot.libraries.dialogs_py.InputDialog method*), 124
register() (*robot.libraries.dialogs_py.MessageDialog method*), 111
register() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 150
register() (*robot.libraries.dialogs_py.PassFailDialog method*), 163
register() (*robot.libraries.dialogs_py.SelectionDialog method*), 137
register() (*robot.output.listenermethods.LibraryListenerMethods method*), 197
register() (*robot.output.listeners.LibraryListeners method*), 198
register() (*robot.utils.connectioncache.ConnectionCache method*), 284
register_console_logger() (*robot.output.logger.Logger method*), 198
register_error_listener() (*robot.output.logger.Logger method*), 199
register_error_listener() (*robot.output.output.Output method*), 201
register_listeners() (*robot.output.logger.Logger method*), 198
register_logger() (*robot.output.logger.Logger method*), 198
register_run_keyword() (*in module robot.libraries.BuiltIn*), 46
register_syslog() (*robot.output.logger.Logger method*), 198
register_xml_logger() (*robot.output.logger.Logger method*), 198
relative_source() (*robot.reporting.jsbuildingcontext.JsBuildingContext method*), 220
release() (*robot.output.pyloggingconf.RobotHandler method*), 202
release() (*robot.running.outputcapture.JavaCapturer method*), 273
release() (*robot.running.outputcapture.PythonCapturer method*), 273
reload_library() (*robot.libraries.BuiltIn.BuiltIn method*), 34
reload_library() (*robot.running.namespace.Namespace method*), 273
Remote (*class in robot.libraries.Remote*), 75
RemoteError, 311
RemoteResult (*class in robot.libraries.Remote*), 75
RemovalMessage (*class in robot.result.keywordremover*), 238
remove() (*robot.model.tags.Tags method*), 183
remove() (*robot.variables.store.VariableStore method*), 301
remove_data_not_needed_in_report() (*robot.reporting.jsexecutionresult.JsExecutionResult*

method), 221
remove_directory() (*robot.libraries.OperatingSystem.OperatingSystem method*), 64
remove_duplicates() (*robot.libraries.Collections.Collections method*), 52
remove_element() (*robot.libraries.XML.XML method*), 102
remove_element_attribute() (*robot.libraries.XML.XML method*), 101
remove_element_attributes() (*robot.libraries.XML.XML method*), 102
remove_elements() (*robot.libraries.XML.XML method*), 102
remove_elements_attribute() (*robot.libraries.XML.XML method*), 102
remove_elements_attributes() (*robot.libraries.XML.XML method*), 102
remove_empty_suites() (*robot.model.testsuite.TestSuite method*), 189
remove_empty_suites() (*robot.result.model.TestSuite method*), 247
remove_empty_suites() (*robot.running.model.TestSuite method*), 271
remove_environment_variable() (*robot.libraries.OperatingSystem.OperatingSystem method*), 66
remove_file() (*robot.libraries.OperatingSystem.OperatingSystem method*), 64
remove_files() (*robot.libraries.OperatingSystem.OperatingSystem method*), 64
remove_from_dictionary() (*robot.libraries.Collections.Collections method*), 53
remove_from_list() (*robot.libraries.Collections.Collections method*), 53
remove_keywords (*robot.conf.settings.RebotSettings attribute*), 18
remove_keywords (*robot.conf.settings.RobotSettings attribute*), 18
remove_keywords() (*robot.result.model.TestSuite method*), 245
remove_path() (*in module robot.pythonpathsetter*), 312
remove_string() (*robot.libraries.String.String method*), 81
remove_string_using_regexp() (*robot.libraries.String.String method*), 81
remove_tags (*robot.model.configurer.SuiteConfigurer attribute*), 170
remove_tags (*robot.result.configurer.SuiteConfigurer attribute*), 228
remove_tags() (*robot.libraries.BuiltIn.BuiltIn method*), 34
remove_values_from_list() (*robot.libraries.Collections.Collections method*), 53
removeFilter() (*robot.output.pyloggingconf.RobotHandler method*), 202
RemoveKeywords (*class in robot.result.resultbuilder*), 248
repeat_keyword() (*robot.libraries.BuiltIn.BuiltIn method*), 34
replace() (*robot.running.arguments.argumentresolver.VariableReplacer method*), 259
replace_defaults() (*robot.running.arguments.argumentmapper.KeywordCallTemplate method*), 258
replace_list() (*robot.variables.replacer.VariableReplacer method*), 299
replace_list() (*robot.variables.scopes.GlobalVariables method*), 300
replace_list() (*robot.variables.scopes.VariableScopes method*), 300
replace_list() (*robot.variables.variables.Variables method*), 302
replace_scalar() (*robot.variables.replacer.VariableReplacer method*), 299
replace_scalar() (*robot.variables.scopes.GlobalVariables method*), 300
replace_scalar() (*robot.variables.scopes.VariableScopes method*), 300
replace_scalar() (*robot.variables.variables.Variables method*), 302
replace_string() (*robot.libraries.String.String method*), 81
replace_string() (*robot.variables.replacer.VariableReplacer method*), 300
replace_string() (*robot.variables.scopes.GlobalVariables method*), 300
replace_string() (*robot.variables.scopes.VariableScopes method*), 300
replace_string() (*robot.variables.variables.Variables method*), 302
replace_string_using_regexp() (*robot.libraries.String.String method*), 81
replace_variables() (*robot.libraries.BuiltIn.BuiltIn method*), 35
replace_variables() (*robot.running.timeouts.KeywordTimeout method*), 260
replace_variables() (*robot.running.timeouts.TestTimeout method*),

260
report (*robot.conf.settings.RebotSettings attribute*), 18
report (*robot.conf.settings.RobotSettings attribute*), 18
report() (*robot.output.console.dotted.StatusReporter method*), 193
report_config (*robot.conf.settings.RebotSettings attribute*), 18
report_error() (*robot.variables.tablesetter.DictVariableTableValue method*), 302
report_error() (*robot.variables.tablesetter.ListVariableTableValue method*), 302
report_error() (*robot.variables.tablesetter.ScalarVariableTableValue method*), 302
report_error() (*robot.variables.tablesetter.VariableTableValueBase method*), 302
report_invalid_syntax() (*robot.model.imports.Import method*), 173
report_invalid_syntax() (*robot.parsing.model.InitFileSettingTable method*), 210
report_invalid_syntax() (*robot.parsing.model.KeywordTable method*), 211
report_invalid_syntax() (*robot.parsing.model.ResourceFile method*), 208
report_invalid_syntax() (*robot.parsing.model.ResourceFileSettingTable method*), 210
report_invalid_syntax() (*robot.parsing.model.TestCase method*), 211
report_invalid_syntax() (*robot.parsing.model.TestCaseFile method*), 208
report_invalid_syntax() (*robot.parsing.model.TestCaseFileSettingTable method*), 209
report_invalid_syntax() (*robot.parsing.model.TestCaseTable method*), 211
report_invalid_syntax() (*robot.parsing.model.TestDataDirectory method*), 209
report_invalid_syntax() (*robot.parsing.model.UserKeyword method*), 212
report_invalid_syntax() (*robot.parsing.model.Variable method*), 211
report_invalid_syntax() (*robot.parsing.model.VariableTable method*), 210
report_invalid_syntax() (*robot.parsing.settings.Arguments method*), 216
report_invalid_syntax() (*robot.parsing.settings.Documentation method*), 214
report_invalid_syntax() (*robot.parsing.settings.Fixture method*), 215
report_invalid_syntax() (*robot.parsing.settings.Library method*), 217
report_invalid_syntax() (*robot.parsing.settings.Metadata method*), 217
report_invalid_syntax() (*robot.parsing.settings.Resource method*), 217
report_invalid_syntax() (*robot.parsing.settings.Return method*), 216
report_invalid_syntax() (*robot.parsing.settings.Setting method*), 214
report_invalid_syntax() (*robot.parsing.settings.Tags method*), 216
report_invalid_syntax() (*robot.parsing.settings.Template method*), 215
report_invalid_syntax() (*robot.parsing.settings.Timeout method*), 215
report_invalid_syntax() (*robot.parsing.settings.Variables method*), 218
report_invalid_syntax() (*robot.running.model.Variable method*), 272
ReportWriter (*class in robot.reporting.logreportwriters*), 222
request() (*robot.libraries.Remote.TimeoutHTTPSTransport method*), 76
request() (*robot.libraries.Remote.TimeoutHTTPTransport method*), 76
required_arguments (*robot.parsing.restsupport.CaptureRobotData attribute*), 214
Reserved (*class in robot.libraries.Reserved*), 77
reset() (*robot.output.console.highlighting.AnsiHighlighter method*), 194
reset() (*robot.output.console.highlighting.DosHighlighter method*), 194
reset() (*robot.output.console.highlighting.NoHighlighting method*), 194
reset() (*robot.parsing.htmlreader.HtmlReader method*), 207
reset() (*robot.parsing.settings.Arguments method*), 216

reset() (*robot.parsing.settings.Documentation method*), 215
reset() (*robot.parsing.settings.Fixture method*), 215
reset() (*robot.parsing.settings.Library method*), 217
reset() (*robot.parsing.settings.Metadata method*), 216
reset() (*robot.parsing.settings.Resource method*), 217
reset() (*robot.parsing.settings.Return method*), 216
reset() (*robot.parsing.settings.Setting method*), 214
reset() (*robot.parsing.settings.Tags method*), 216
reset() (*robot.parsing.settings.Template method*), 215
reset() (*robot.parsing.settings.Timeout method*), 215
reset() (*robot.parsing.settings.Variables method*), 218
reset() (*robot.running.importer.Importer method*), 263
reset_count() (*robot.output.console.verbose.KeywordMarker method*), 195
resizable() (*robot.libraries.dialogs_py.InputDialog method*), 124
resizable() (*robot.libraries.dialogs_py.MessageDialog method*), 111
resizable() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 150
resizable() (*robot.libraries.dialogs_py.PassFailDialog method*), 163
resizable() (*robot.libraries.dialogs_py.SelectionDialog method*), 137
resolve() (*robot.running.arguments.argumentmapper.DefaultValue method*), 258
resolve() (*robot.running.arguments.argumentresolver.ArgumentResolver method*), 259
resolve() (*robot.running.arguments.argumentresolver.NamedArgumentResolver method*), 259
resolve() (*robot.running.arguments.argumentresolver.NullNamedArgumentResolver method*), 259
resolve() (*robot.running.arguments.argumentspec.ArgumentSpec method*), 259
resolve() (*robot.variables.assigner.NoReturnValueResolver method*), 298
resolve() (*robot.variables.assigner.OneReturnValueResolver method*), 298
resolve() (*robot.variables.assigner.ScalarsAndListReturnValueResolver method*), 298
resolve() (*robot.variables.assigner.ScalarsOnlyReturnValueResolver method*), 298
resolve() (*robot.variables.tablesetter.DictVariableTableValue method*), 302
resolve() (*robot.variables.tablesetter.ListVariableTableValue method*), 302
resolve() (*robot.variables.tablesetter.ScalarVariableTableValue method*), 302
resolve() (*robot.variables.tablesetter.VariableTableValueBase method*), 302
resolve_alias_or_index() (*robot.utils.connectioncache.ConnectionCache method*), 285
resolve_delayed() (*robot.variables.scopes.GlobalVariables method*), 300
resolve_delayed() (*robot.variables.scopes.VariableScopes method*), 300
resolve_delayed() (*robot.variables.store.VariableStore method*), 301
resolve_delayed() (*robot.variables.variables.Variables method*), 302
resolve_delayed_message() (*robot.output.loggerhelper.Message method*), 200
Resource (*class in robot.parsing.settings*), 217
resource (*robot.running.model.TestSuite attribute*), 269
resource() (*robot.model.imports.Imports method*), 173
RESOURCE_FILE_TYPE (*robot.running.handlerstore.HandlerStore attribute*), 263
RESOURCE_FILE_TYPE (*robot.running.userkeyword.UserLibrary attribute*), 278
ResourceDocBuilder (*class in robot.libdocpkg.robotbuilder*), 22
ResourceFile (*class in robot.parsing.model*), 208
ResourceFile (*class in robot.running.model*), 272
ResourceFileBuilder (*class in robot.parsing.model*), 262
ResourceFileSettingTable (*class in robot.parsing.model*), 209
RestReader() (*in module robot.parsing.restreader*), 213
Result (*class in robot.result.executionresult*), 229
result (*robot.reporting.resultwriter.Results attribute*), 224
result_config (*robot.libraries.Process.ProcessConfiguration attribute*), 75
ResultFileConfig (*class in robot.reporting.resultwriter*), 224
ResultVisitor (*class in robot.result.visitor*), 251
ResultWriter (*class in robot.reporting.resultwriter*), 224
Return (*class in robot.parsing.settings*), 216
return_code (*robot.result.executionresult.CombinedResult attribute*), 231
return_code (*robot.result.executionresult.Result attribute*), 230
return_from_keyword() (*robot.libraries.BuiltIn.BuiltIn method*), 35

return_from_keyword_if() (*robot.libraries.BuiltIn.BuiltIn method*), 35
ReturnFromKeyword, 310
ReturnValueResolver() (*in module robot.variables.assigner*), 297
reverse_list() (*robot.libraries.Collections.Collections method*), 53
robot (*module*), 9
robot.api (*module*), 7, 12
robot.api.deco (*module*), 12
robot.api.logger (*module*), 13
robot.conf (*module*), 15
robot.conf.gatherfailed (*module*), 15
robot.conf.settings (*module*), 17
robot.errors (*module*), 306
robot.htmldata (*module*), 19
robot.htmldata.htmlfilewriter (*module*), 19
robot.htmldata.jsonwriter (*module*), 20
robot.htmldata.normaltemplate (*module*), 20
robot.htmldata.template (*module*), 21
robot.libdoc (*module*), 311
robot.libdocpkg (*module*), 21
robot.libdocpkg.builder (*module*), 21
robot.libdocpkg.consoleviewer (*module*), 21
robot.libdocpkg.htmlwriter (*module*), 21
robot.libdocpkg.javabuilder (*module*), 22
robot.libdocpkg.model (*module*), 22
robot.libdocpkg.output (*module*), 22
robot.libdocpkg.robotbuilder (*module*), 22
robot.libdocpkg.specbuilder (*module*), 23
robot.libdocpkg.writer (*module*), 23
robot.libdocpkg.xmlwriter (*module*), 23
robot.libraries (*module*), 23
robot.libraries.BuiltIn (*module*), 23
robot.libraries.Collections (*module*), 47
robot.libraries.DateTime (*module*), 53
robot.libraries.Dialogs (*module*), 58
robot.libraries.dialogs_py (*module*), 104
robot.libraries.Easter (*module*), 59
robot.libraries.OperatingSystem (*module*), 59
robot.libraries.Process (*module*), 69
robot.libraries.Remote (*module*), 75
robot.libraries.Reserved (*module*), 77
robot.libraries.Screenshot (*module*), 77
robot.libraries.String (*module*), 78
robot.libraries.Telnet (*module*), 84
robot.libraries.XML (*module*), 93
robot.model (*module*), 169
robot.model.configurer (*module*), 170
robot.model.criticality (*module*), 171
robot.model.filter (*module*), 171
robot.model.imports (*module*), 173
robot.model.itemlist (*module*), 174
robot.model.keyword (*module*), 174
robot.model.message (*module*), 176
robot.model.metadata (*module*), 177
robot.model.modelobject (*module*), 178
robot.model.modifier (*module*), 178
robot.model.namepatterns (*module*), 179
robot.model.statistics (*module*), 179
robot.model.stats (*module*), 181
robot.model.suitestatistics (*module*), 183
robot.model.tags (*module*), 183
robot.model.tagsetter (*module*), 184
robot.model.tagstatistics (*module*), 185
robot.model.testcase (*module*), 186
robot.model.testsuite (*module*), 187
robot.model.totalstatistics (*module*), 189
robot.model.visitor (*module*), 190
robot.output (*module*), 192
robot.output.console (*module*), 192
robot.output.console.dotted (*module*), 192
robot.output.console.highlighting (*module*), 194
robot.output.console.quiet (*module*), 194
robot.output.console.verbose (*module*), 195
robot.output.debugfile (*module*), 195
robot.output.filelogger (*module*), 195
robot.output.librarylogger (*module*), 196
robot.output.listenerarguments (*module*), 196
robot.output.listenermethods (*module*), 197
robot.output.listeners (*module*), 198
robot.output.logger (*module*), 198
robot.output.loggerhelper (*module*), 199
robot.output.output (*module*), 201
robot.output.pyloggingconf (*module*), 201
robot.output.stdoutlogsplitter (*module*), 203
robot.output.xmllogger (*module*), 203
robot.parsing (*module*), 204
robot.parsing.comments (*module*), 205
robot.parsing.datarow (*module*), 205
robot.parsing.htmlreader (*module*), 206
robot.parsing.model (*module*), 207
robot.parsing.populators (*module*), 213
robot.parsing.restreader (*module*), 213
robot.parsing.restsupport (*module*), 213
robot.parsing.settings (*module*), 214
robot.parsing.tablepopulators (*module*), 218
robot.parsing.tsvreader (*module*), 220
robot.parsing.txtreader (*module*), 220
robot.pythonpathsetter (*module*), 312
robot.rebot (*module*), 312
robot.reporting (*module*), 220

robot.reporting.jsbuildingcontext (*module*), 220
robot.reporting.jsexecutionresult (*module*), 221
robot.reporting.jsmodelbuilders (*module*), 221
robot.reporting.jswriter (*module*), 222
robot.reporting.logreportwriters (*module*), 222
robot.reporting.outputwriter (*module*), 222
robot.reporting.resultwriter (*module*), 224
robot.reporting.stringcache (*module*), 224
robot.reporting.xunitwriter (*module*), 225
robot.result (*module*), 227
robot.result.configurer (*module*), 228
robot.result.executionerrors (*module*), 229
robot.result.executionresult (*module*), 229
robot.result.flattenkeywordmatcher (*module*), 232
robot.result.keywordremover (*module*), 232
robot.result.merger (*module*), 238
robot.result.messagefilter (*module*), 239
robot.result.model (*module*), 240
robot.result.resultbuilder (*module*), 247
robot.result.suiteteardownfailed (*module*), 249
robot.result.visitor (*module*), 251
robot.result.xmlelementhandlers (*module*), 253
robot.run (*module*), 314
robot.running (*module*), 256
robot.running.arguments (*module*), 258
robot.running.arguments.argumentmapper (*module*), 258
robot.running.arguments.argumentparser (*module*), 258
robot.running.arguments.argumentresolver (*module*), 259
robot.running.arguments.argumentspec (*module*), 259
robot.running.arguments.argumentvalidator (*module*), 260
robot.running.arguments.embedded (*module*), 260
robot.running.builder (*module*), 261
robot.running.context (*module*), 262
robot.running.defaults (*module*), 262
robot.running.dynamicmethods (*module*), 262
robot.running.handlers (*module*), 263
robot.running.handlerstore (*module*), 263
robot.running.importer (*module*), 263
robot.running.librarykeywordrunner (*module*), 264
robot.running.libraryscopes (*module*), 264
robot.running.model (*module*), 265
robot.running.namespace (*module*), 272
robot.running.outputcapture (*module*), 273
robot.running.randomizer (*module*), 273
robot.running.runkwregister (*module*), 274
robot.running.runner (*module*), 274
robot.running.signalhandler (*module*), 275
robot.running.status (*module*), 276
robot.running.statusreporter (*module*), 277
robot.running.steprunner (*module*), 277
robot.running.testlibraries (*module*), 278
robot.running.timeouts (*module*), 260
robot.running.timeouts.posix (*module*), 261
robot.running.timeouts.windows (*module*), 261
robot.running.usererrorhandler (*module*), 278
robot.running.userkeyword (*module*), 278
robot.running.userkeywordrunner (*module*), 279
robot.testdoc (*module*), 316
robot.tidy (*module*), 317
robot.utils (*module*), 279
robot.utils.application (*module*), 280
robot.utils.argumentparser (*module*), 280
robot.utils.asserts (*module*), 281
robot.utils.charwidth (*module*), 283
robot.utils.compat (*module*), 284
robot.utils.compress (*module*), 284
robot.utils.connectioncache (*module*), 284
robot.utils.dotdict (*module*), 285
robot.utils.encoding (*module*), 286
robot.utils.encodingsniffer (*module*), 286
robot.utils.error (*module*), 286
robot.utils.escaping (*module*), 287
robot.utils.etreewrapper (*module*), 287
robot.utils.frange (*module*), 287
robot.utils.htmlformatters (*module*), 287
robot.utils.importer (*module*), 289
robot.utils.markuputils (*module*), 290
robot.utils.markupwriters (*module*), 290
robot.utils.match (*module*), 291
robot.utils.misc (*module*), 291
robot.utils.normalizing (*module*), 292
robot.utils.platform (*module*), 293
robot.utils.recommendations (*module*), 293
robot.utils.robotenv (*module*), 293
robot.utils.robotinspect (*module*), 293
robot.utils.robotio (*module*), 293
robot.utils.robotpath (*module*), 293
robot.utils.robottime (*module*), 294
robot.utils.robottypes (*module*), 295
robot.utils.robottypes2 (*module*), 295
robot.utils.setter (*module*), 296

robot.utils.sortable (*module*), 296
robot.utils.text (*module*), 296
robot.utils.unic (*module*), 296
robot.utils.utf8reader (*module*), 297
robot.variables (*module*), 297
robot.variables.assigner (*module*), 297
robot.variables.filesetter (*module*), 298
robot.variables.finders (*module*), 298
robot.variables.isvar (*module*), 299
robot.variables.notfound (*module*), 299
robot.variables.replacer (*module*), 299
robot.variables.scopes (*module*), 300
robot.variables.splitter (*module*), 301
robot.variables.store (*module*), 301
robot.variables.tablesetter (*module*), 302
robot.variables.variables (*module*), 302
robot.version (*module*), 318
robot.writer (*module*), 303
robot.writer.aligners (*module*), 303
robot.writer.dataextractor (*module*), 303
robot.writer.datafilewriter (*module*), 303
robot.writer.filewriters (*module*), 305
robot.writer.formatters (*module*), 305
robot.writer.htmlformatter (*module*), 305
robot.writer.htmltemplate (*module*), 306
robot.writer.rowsplitter (*module*), 306
robot_format (*robot.writer.datafilewriter.WritingContext attribute*), 304
robot_handler_enabled() (*in module robot.output.pyloggingconf*), 201
ROBOT_LIBRARY_SCOPE (*robot.libraries.BuiltIn.BuiltIn attribute*), 26
ROBOT_LIBRARY_SCOPE (*robot.libraries.Collections.Collections attribute*), 48
ROBOT_LIBRARY_SCOPE (*robot.libraries.OperatingSystem.OperatingSystem attribute*), 60
ROBOT_LIBRARY_SCOPE (*robot.libraries.Process.Process attribute*), 71
ROBOT_LIBRARY_SCOPE (*robot.libraries.Remote.Remote attribute*), 75
ROBOT_LIBRARY_SCOPE (*robot.libraries.Reserved.Reserved attribute*), 77
ROBOT_LIBRARY_SCOPE (*robot.libraries.Screenshot.Screenshot attribute*), 78
ROBOT_LIBRARY_SCOPE (*robot.libraries.String.String attribute*), 79
ROBOT_LIBRARY_SCOPE (*robot.libraries.Telnet.Telnet attribute*), 87
ROBOT_LIBRARY_SCOPE (*robot.libraries.XML.XML attribute*), 97
ROBOT_LIBRARY_VERSION (*robot.libraries.BuiltIn.BuiltIn attribute*), 26
ROBOT_LIBRARY_VERSION (*robot.libraries.Collections.Collections attribute*), 48
ROBOT_LIBRARY_VERSION (*robot.libraries.OperatingSystem.OperatingSystem attribute*), 60
ROBOT_LIBRARY_VERSION (*robot.libraries.Process.Process attribute*), 71
ROBOT_LIBRARY_VERSION (*robot.libraries.Screenshot.Screenshot attribute*), 78
ROBOT_LIBRARY_VERSION (*robot.libraries.String.String attribute*), 79
ROBOT_LIBRARY_VERSION (*robot.libraries.Telnet.Telnet attribute*), 87
ROBOT_LIBRARY_VERSION (*robot.libraries.XML.XML attribute*), 97
ROBOT_SUPPRESS_NAME (*robot.libraries.Telnet.NoMatchError attribute*), 93
RobotDataStorage (*class in robot.parsing.restsupport*), 214
RobotError, 306
RobotFramework (*class in robot.run*), 314
RobotHandler (*class in robot.output.pyloggingconf*), 201
RobotHandler (*class in robot.result.xmlelementhandlers*), 253
RobotModelWriter (*class in robot.reporting.logreportwriters*), 222
RobotNotRunningError, 46
RobotSettings (*class in robot.conf.settings*), 17
RootHandler (*class in robot.result.xmlelementhandlers*), 253
RootSuiteHandler (*class in robot.result.xmlelementhandlers*), 253
roundup() (*in module robot.utils.misc*), 291
rowconfigure() (*robot.libraries.dialogs_py.InputDialog method*), 124
rowconfigure() (*robot.libraries.dialogs_py.MessageDialog method*), 111
rowconfigure() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 150
rowconfigure() (*robot.libraries.dialogs_py.PassFailDialog method*), 163
rowconfigure() (*robot.libraries.dialogs_py.SelectionDialog method*), 137

rows_from_table() (*robot.writer.dataextractor.DataExtractor method*), 303
RowSplitter (*class in robot.writer.rowsplitter*), 306
rpa (*robot.conf.settings.RebotSettings attribute*), 18
rpa (*robot.conf.settings.RobotSettings attribute*), 18
rpa (*robot.model.testsuite.TestSuite attribute*), 187
rpa (*robot.result.model.TestSuite attribute*), 247
rpa (*robot.running.model.TestSuite attribute*), 271
RulerFormatter (*class in robot.utils.htmlformatters*), 288
run() (*in module robot*), 9
run() (*in module robot.run*), 315
run() (*robot.libraries.OperatingSystem.OperatingSystem method*), 60
run() (*robot.parsing.restsupport.CaptureRobotData method*), 213
run() (*robot.running.librarykeywordrunner.EmbeddedArgumentsRunner method*), 264
run() (*robot.running.librarykeywordrunner.LibraryKeywordRunner method*), 264
run() (*robot.running.librarykeywordrunner.RunKeywordRunner method*), 264
run() (*robot.running.model.ForLoop method*), 268
run() (*robot.running.model.Keyword method*), 265
run() (*robot.running.model.TestSuite method*), 270
run() (*robot.running.steprunner.ForInEnumerateRunner method*), 277
run() (*robot.running.steprunner.ForInRangeRunner method*), 277
run() (*robot.running.steprunner.ForInRunner method*), 277
run() (*robot.running.steprunner.ForInZipRunner method*), 277
run() (*robot.running.steprunner.InvalidForRunner method*), 278
run() (*robot.running.timeouts.KeywordTimeout method*), 260
run() (*robot.running.timeouts.TestTimeout method*), 260
run() (*robot.running.usererrorhandler.UserErrorHandler method*), 278
run() (*robot.running.userkeywordrunner.EmbeddedArgumentsRunner method*), 279
run() (*robot.running.userkeywordrunner.UserKeywordRunner method*), 279
run_and_return_rc() (*robot.libraries.OperatingSystem.OperatingSystem method*), 61
run_and_return_rc_and_output() (*robot.libraries.OperatingSystem.OperatingSystem method*), 61
run_cli() (*in module robot*), 10
run_cli() (*in module robot.run*), 314
run_empty_suite (*robot.conf.settings.RobotSettings attribute*), 17
run_keyword() (*robot.libraries.BuiltIn.BuiltIn method*), 35
run_keyword() (*robot.libraries.Remote.Remote method*), 75
run_keyword() (*robot.libraries.Remote.XmlRpcRemoteClient method*), 76
run_keyword() (*robot.libraries.Reserved.Reserved method*), 77
run_keyword_and_continue_on_failure() (*robot.libraries.BuiltIn.BuiltIn method*), 35
run_keyword_and_expect_error() (*robot.libraries.BuiltIn.BuiltIn method*), 35
run_keyword_and_ignore_error() (*robot.libraries.BuiltIn.BuiltIn method*), 36
run_keyword_and_return() (*robot.libraries.BuiltIn.BuiltIn method*), 36
run_keyword_and_return_if() (*robot.libraries.BuiltIn.BuiltIn method*), 36
run_keyword_and_return_status() (*robot.libraries.BuiltIn.BuiltIn method*), 36
run_keyword_if() (*robot.libraries.BuiltIn.BuiltIn method*), 36
run_keyword_if_all_critical_tests_passed() (*robot.libraries.BuiltIn.BuiltIn method*), 37
run_keyword_if_all_tests_passed() (*robot.libraries.BuiltIn.BuiltIn method*), 37
run_keyword_if_any_critical_tests_failed() (*robot.libraries.BuiltIn.BuiltIn method*), 37
run_keyword_if_any_tests_failed() (*robot.libraries.BuiltIn.BuiltIn method*), 37
run_keyword_if_test_failed() (*robot.libraries.BuiltIn.BuiltIn method*), 37
run_keyword_if_test_passed() (*robot.libraries.BuiltIn.BuiltIn method*), 37
run_keyword_if_timeout_occurred() (*robot.libraries.BuiltIn.BuiltIn method*), 38
run_keyword_unless() (*robot.libraries.BuiltIn.BuiltIn method*), 38
run_keyword_variant() (*in module robot.libraries.BuiltIn*), 23
run_keywords() (*robot.libraries.BuiltIn.BuiltIn*

method), 38
run_process() (*robot.libraries.Process.Process method*), 72
run_step() (*robot.running.steprunner.StepRunner method*), 277
run_steps() (*robot.running.steprunner.StepRunner method*), 277
RunKeyword (*class in robot.running.dynamicmethods*), 262
RunKeywordRunner (*class in robot.running.librarykeywordrunner*), 264
Runner (*class in robot.running.runner*), 274

S

save() (*robot.libdocpkg.model.LibraryDoc method*), 22
save() (*robot.parsing.model.ResourceFile method*), 208
save() (*robot.parsing.model.TestCaseFile method*), 208
save() (*robot.parsing.model.TestDataDirectory method*), 209
save() (*robot.result.executionresult.CombinedResult method*), 231
save() (*robot.result.executionresult.Result method*), 230
save_xml() (*robot.libraries.XML.XML method*), 103
ScalarsAndListReturnValueResolver (*class in robot.variables.assigner*), 298
ScalarsOnlyReturnValueResolver (*class in robot.variables.assigner*), 298
ScalarVariableTableValue (*class in robot.variables.tablesetter*), 302
Screenshot (*class in robot.libraries.Screenshot*), 77
ScreenshotTaker (*class in robot.libraries.Screenshot*), 78
search() (*robot.libdocpkg.consoleviewer.KeywordMatcher method*), 21
secs_to_timestamp() (*in module robot.utils.robottime*), 295
secs_to_timestr() (*in module robot.utils.robottime*), 294
selection_clear() (*robot.libraries.dialogs_py.InputDialog method*), 124
selection_clear() (*robot.libraries.dialogs_py.MessageDialog method*), 111
selection_clear() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 150
selection_clear() (*robot.libraries.dialogs_py.PassFailDialog method*), 163
selection_clear() (*robot.libraries.dialogs_py.SelectionDialog method*), 137
selection_get() (*robot.libraries.dialogs_py.InputDialog method*), 124
selection_get() (*robot.libraries.dialogs_py.MessageDialog method*), 111
selection_get() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 150
selection_get() (*robot.libraries.dialogs_py.PassFailDialog method*), 163
selection_get() (*robot.libraries.dialogs_py.SelectionDialog method*), 137
selection_handle() (*robot.libraries.dialogs_py.InputDialog method*), 124
selection_handle() (*robot.libraries.dialogs_py.MessageDialog method*), 111
selection_handle() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 150
selection_handle() (*robot.libraries.dialogs_py.PassFailDialog method*), 163
selection_handle() (*robot.libraries.dialogs_py.SelectionDialog method*), 137
selection_own() (*robot.libraries.dialogs_py.InputDialog method*), 124
selection_own() (*robot.libraries.dialogs_py.MessageDialog method*), 111
selection_own() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 150
selection_own() (*robot.libraries.dialogs_py.PassFailDialog method*), 163
selection_own() (*robot.libraries.dialogs_py.SelectionDialog method*), 137
selection_own_get() (*robot.libraries.dialogs_py.InputDialog method*), 124
selection_own_get() (*robot.libraries.dialogs_py.MessageDialog method*), 111
selection_own_get() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 150
selection_own_get() (*robot.libraries.dialogs_py.PassFailDialog method*), 163
selection_own_get() (*robot.libraries.dialogs_py.SelectionDialog method*), 137
SelectionDialog (*class in*

robot.libraries.dialogs_py), 130
send() (*robot.libraries.dialogs_py.InputDialog method*), 124
send() (*robot.libraries.dialogs_py.MessageDialog method*), 111
send() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 150
send() (*robot.libraries.dialogs_py.PassFailDialog method*), 163
send() (*robot.libraries.dialogs_py.SelectionDialog method*), 137
send_content() (*robot.libraries.Remote.TimeoutHTTPSTransport method*), 76
send_content() (*robot.libraries.Remote.TimeoutHTTPTransport method*), 76
send_host() (*robot.libraries.Remote.TimeoutHTTPSTransport method*), 76
send_host() (*robot.libraries.Remote.TimeoutHTTPTransport method*), 76
send_request() (*robot.libraries.Remote.TimeoutHTTPSTransport method*), 76
send_request() (*robot.libraries.Remote.TimeoutHTTPTransport method*), 76
send_signal_to_process() (*robot.libraries.Process.Process method*), 73
send_user_agent() (*robot.libraries.Remote.TimeoutHTTPSTransport method*), 76
send_user_agent() (*robot.libraries.Remote.TimeoutHTTPTransport method*), 76
seq2str() (*in module robot.utils.misc*), 292
seq2str2() (*in module robot.utils.misc*), 292
set() (*robot.result.keywordremover.RemovalMessage method*), 238
set() (*robot.variables.filesetter.VariableFileSetter method*), 298
set() (*robot.variables.tablesetter.VariableTableSetter method*), 302
set_cdata_mode() (*robot.parsing.htmlreader.HtmlReader method*), 207
set_criticality() (*robot.result.model.TestSuite method*), 245
set_debuglevel() (*robot.libraries.Telnet.TelnetConnection method*), 92
set_default_log_level() (*robot.libraries.Telnet.TelnetConnection method*), 89
set_earlier_failures() (*robot.errors.ContinueForLoop method*), 310
set_earlier_failures() (*robot.errors.ExecutionPassed method*), 309
set_earlier_failures() (*robot.errors.ExitForLoop method*), 310
set_earlier_failures() (*robot.errors.PassExecution method*), 309
set_earlier_failures() (*robot.errors.ReturnFromKeyword method*), 310
set_element_attribute() (*robot.libraries.XML.XML method*), 101
set_element_tag() (*robot.libraries.XML.XML method*), 101
set_element_text() (*robot.libraries.XML.XML method*), 101
set_elements_attribute() (*robot.libraries.XML.XML method*), 101
set_elements_tag() (*robot.libraries.XML.XML method*), 101
set_elements_text() (*robot.libraries.XML.XML method*), 101
set_encoding() (*robot.libraries.Telnet.TelnetConnection method*), 88
set_env_var() (*in module robot.utils.robotenv*), 293
set_environment_variable() (*robot.libraries.OperatingSystem.OperatingSystem method*), 66
set_execution_mode() (*robot.result.executionresult.CombinedResult method*), 231
set_execution_mode() (*robot.result.executionresult.Result method*), 230
set_from_file() (*robot.variables.scopes.GlobalVariables method*), 300
set_from_file() (*robot.variables.scopes.VariableScopes method*), 300
set_from_file() (*robot.variables.variables.Variables method*), 302
set_from_variable_table() (*robot.variables.scopes.GlobalVariables method*), 301
set_from_variable_table() (*robot.variables.scopes.VariableScopes method*), 300
set_from_variable_table() (*robot.variables.variables.Variables method*), 302
set_global() (*robot.variables.scopes.SetVariables method*), 301
set_global() (*robot.variables.scopes.VariableScopes method*), 300
set_global_variable() (*robot.libraries.BuiltIn.BuiltIn method*), 38

set_header() (*robot.parsing.model.InitFileSettingTable method*), 210
set_header() (*robot.parsing.model.KeywordTable method*), 211
set_header() (*robot.parsing.model.ResourceFileSettingTable method*), 210
set_header() (*robot.parsing.model.TestCaseFileSettingTable method*), 209
set_header() (*robot.parsing.model.TestCaseTable method*), 210
set_header() (*robot.parsing.model.VariableTable method*), 210
set_if_removed() (*robot.result.keywordremover.RemovalMessage method*), 238
set_keyword() (*robot.variables.scopes.SetVariables method*), 301
set_keyword() (*robot.variables.scopes.VariableScopes method*), 300
set_keyword_timeout() (*robot.running.timeouts.TestTimeout method*), 260
set_level() (*in module robot.output.pyloggingconf*), 201
set_level() (*robot.output.filelogger.FileLogger method*), 196
set_level() (*robot.output.logger.Logger method*), 199
set_level() (*robot.output.loggerhelper.AbstractLogger method*), 199
set_level() (*robot.output.loggerhelper.IsLogged method*), 200
set_level() (*robot.output.output.Output method*), 201
set_library_search_order() (*robot.libraries.BuiltIn.BuiltIn method*), 38
set_list_value() (*robot.libraries.Collections.Collections method*), 53
set_log_level() (*robot.libraries.BuiltIn.BuiltIn method*), 39
set_log_level() (*robot.output.listeners.LibraryListeners method*), 198
set_log_level() (*robot.output.listeners.Listeners method*), 198
set_log_level() (*robot.output.output.Output method*), 201
set_log_level() (*robot.output.xmllogger.XmlLogger method*), 203
set_log_level() (*robot.reporting.outputwriter.OutputWriter method*), 223
set_modified_time() (*robot.libraries.OperatingSystem.OperatingSystem method*), 68
set_name() (*robot.output.pyloggingconf.RobotHandler method*), 202
set_newline() (*robot.libraries.Telnet.TelnetConnection method*), 88
set_option_negotiation_callback() (*robot.libraries.Telnet.TelnetConnection method*), 92
set_prompt() (*robot.libraries.Telnet.TelnetConnection method*), 88
set_screenshot_directory() (*robot.libraries.Screenshot.Screenshot method*), 78
set_search_order() (*robot.running.namespace.Namespace method*), 273
set_suite() (*robot.variables.scopes.SetVariables method*), 301
set_suite() (*robot.variables.scopes.VariableScopes method*), 300
set_suite_documentation() (*robot.libraries.BuiltIn.BuiltIn method*), 39
set_suite_metadata() (*robot.libraries.BuiltIn.BuiltIn method*), 39
set_suite_variable() (*robot.libraries.BuiltIn.BuiltIn method*), 39
set_tags() (*robot.libraries.BuiltIn.BuiltIn method*), 40
set_tags() (*robot.model.testsuite.TestSuite method*), 188
set_tags() (*robot.result.model.TestSuite method*), 247
set_tags() (*robot.running.model.TestSuite method*), 271
set_task_variable() (*robot.libraries.BuiltIn.BuiltIn method*), 40
set_telnetlib_log_level() (*robot.libraries.Telnet.TelnetConnection method*), 89
set_test() (*robot.variables.scopes.SetVariables method*), 301
set_test() (*robot.variables.scopes.VariableScopes method*), 300
set_test_documentation() (*robot.libraries.BuiltIn.BuiltIn method*), 40
set_test_message() (*robot.libraries.BuiltIn.BuiltIn method*), 40
set_test_variable() (*robot.libraries.BuiltIn.BuiltIn method*), 40

set_timeout() (*robot.libraries.Telnet.TelnetConnection method*), 88
set_to_dictionary() (*robot.libraries.Collections.Collections method*), 53
set_variable() (*robot.libraries.BuiltIn.BuiltIn method*), 40
set_variable_if() (*robot.libraries.BuiltIn.BuiltIn method*), 41
setdefault() (*robot.model.metadata.Metadata method*), 177
setdefault() (*robot.utils.dotdict.DotDict method*), 285
setdefault() (*robot.utils.normalizing.NormalizedDict method*), 292
setFormatter() (*robot.output.pyloggingconf.RobotHandler method*), 202
setLevel() (*robot.output.pyloggingconf.RobotHandler method*), 202
setter (*class in robot.utils.setter*), 296
SetterAwareType (*class in robot.utils.setter*), 296
Setting (*class in robot.parsing.settings*), 214
setting_name (*robot.parsing.settings.Metadata attribute*), 216
SettingPopulator (*class in robot.parsing.tablepopulators*), 219
settings (*robot.parsing.model.TestCase attribute*), 211
settings (*robot.parsing.model.UserKeyword attribute*), 211
SettingTablePopulator (*class in robot.parsing.tablepopulators*), 218
setup (*robot.model.keyword.Keywords attribute*), 176
setup_executed() (*robot.running.status.SuiteStatus method*), 276
setup_executed() (*robot.running.status.TestStatus method*), 276
setup_message (*robot.running.status.ParentMessage attribute*), 277
setup_message (*robot.running.status.SuiteMessage attribute*), 277
setup_message (*robot.running.status.TestMessage attribute*), 276
SETUP_TYPE (*robot.model.keyword.Keyword attribute*), 174
SETUP_TYPE (*robot.result.model.Keyword attribute*), 242
SETUP_TYPE (*robot.running.model.ForLoop attribute*), 267
SETUP_TYPE (*robot.running.model.Keyword attribute*), 266
setvar() (*robot.libraries.dialogs_py.InputDialog method*), 124
setvar() (*robot.libraries.dialogs_py.MessageDialog method*), 111
setvar() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 150
setvar() (*robot.libraries.dialogs_py.PassFailDialog method*), 163
setvar() (*robot.libraries.dialogs_py.SelectionDialog method*), 137
SetVariables (*class in robot.variables.scopes*), 301
severe() (*robot.parsing.restsupport.CaptureRobotData method*), 214
shortdoc (*robot.libdocpkg.model.KeywordDoc attribute*), 22
shortdoc (*robot.running.usererrorhandler.UserErrorHandler attribute*), 278
shortdoc (*robot.running.userkeyword.EmbeddedArgumentsHandler attribute*), 279
shortdoc (*robot.running.userkeyword.UserKeywordHandler attribute*), 278
should_be_byte_string() (*robot.libraries.String.String method*), 83
should_be_empty() (*robot.libraries.BuiltIn.BuiltIn method*), 41
should_be_equal() (*robot.libraries.BuiltIn.BuiltIn method*), 41
should_be_equal_as_integers() (*robot.libraries.BuiltIn.BuiltIn method*), 41
should_be_equal_as_numbers() (*robot.libraries.BuiltIn.BuiltIn method*), 41
should_be_equal_as_strings() (*robot.libraries.BuiltIn.BuiltIn method*), 42
should_be_lowercase() (*robot.libraries.String.String method*), 83
should_be_string() (*robot.libraries.String.String method*), 82
should_be_titlecase() (*robot.libraries.String.String method*), 83
should_be_true() (*robot.libraries.BuiltIn.BuiltIn method*), 42
should_be_unicode_string() (*robot.libraries.String.String method*), 83
should_be_uppercase() (*robot.libraries.String.String method*), 83
should_contain() (*robot.libraries.BuiltIn.BuiltIn method*), 42
should_contain_any() (*robot.libraries.BuiltIn.BuiltIn method*), 42
should_contain_match() (*robot.libraries.Collections.Collections method*), 48
should_contain_x_times()

(*robot.libraries.BuiltIn.BuiltIn method*), 43
should_end_with() (*robot.libraries.BuiltIn.BuiltIn method*), 43
should_exist() (*robot.libraries.OperatingSystem.OperatingSystem method*), 62
should_match() (*robot.libraries.BuiltIn.BuiltIn method*), 43
should_match_regexp() (*robot.libraries.BuiltIn.BuiltIn method*), 43
should_not_be_empty() (*robot.libraries.BuiltIn.BuiltIn method*), 43
should_not_be_equal() (*robot.libraries.BuiltIn.BuiltIn method*), 44
should_not_be_equal_as_integers() (*robot.libraries.BuiltIn.BuiltIn method*), 44
should_not_be_equal_as_numbers() (*robot.libraries.BuiltIn.BuiltIn method*), 44
should_not_be_equal_as_strings() (*robot.libraries.BuiltIn.BuiltIn method*), 44
should_not_be_string() (*robot.libraries.String.String method*), 83
should_not_be_true() (*robot.libraries.BuiltIn.BuiltIn method*), 44
should_not_contain() (*robot.libraries.BuiltIn.BuiltIn method*), 44
should_not_contain_any() (*robot.libraries.BuiltIn.BuiltIn method*), 44
should_not_contain_match() (*robot.libraries.Collections.Collections method*), 48
should_not_end_with() (*robot.libraries.BuiltIn.BuiltIn method*), 45
should_not_exist() (*robot.libraries.OperatingSystem.OperatingSystem method*), 62
should_not_match() (*robot.libraries.BuiltIn.BuiltIn method*), 45
should_not_match_regexp() (*robot.libraries.BuiltIn.BuiltIn method*), 45
should_not_start_with() (*robot.libraries.BuiltIn.BuiltIn method*), 45
should_start_with() (*robot.libraries.BuiltIn.BuiltIn method*), 45
shutdown() (*robot.libdocpkg.consoleviewer.ConsoleViewer method*), 21
show() (*robot.libraries.dialogs_py.InputDialog method*), 124
show() (*robot.libraries.dialogs_py.MessageDialog method*), 111
show() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 150
show() (*robot.libraries.dialogs_py.PassFailDialog method*), 163
show() (*robot.libraries.dialogs_py.SelectionDialog method*), 137
single_request() (*robot.libraries.Remote.TimeoutHTTPSTransport method*), 76
single_request() (*robot.libraries.Remote.TimeoutHTTPTransport method*), 76
SingleTagPattern (*class in robot.model.tags*), 183
size() (*robot.libraries.dialogs_py.InputDialog method*), 125
size() (*robot.libraries.dialogs_py.MessageDialog method*), 111
size() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 151
size() (*robot.libraries.dialogs_py.PassFailDialog method*), 164
size() (*robot.libraries.dialogs_py.SelectionDialog method*), 138
sizefrom() (*robot.libraries.dialogs_py.InputDialog method*), 125
sizefrom() (*robot.libraries.dialogs_py.MessageDialog method*), 112
sizefrom() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 151
sizefrom() (*robot.libraries.dialogs_py.PassFailDialog method*), 164
sizefrom() (*robot.libraries.dialogs_py.SelectionDialog method*), 138
skip_teardown_on_exit (*robot.conf.settings.RobotSettings attribute*), 17
slaves() (*robot.libraries.dialogs_py.InputDialog method*), 125
slaves() (*robot.libraries.dialogs_py.MessageDialog method*), 112
slaves() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 151
slaves() (*robot.libraries.dialogs_py.PassFailDialog method*), 164
slaves() (*robot.libraries.dialogs_py.SelectionDialog method*), 138
sleep() (*robot.libraries.BuiltIn.BuiltIn method*), 45

sock_avail() (*robot.libraries.Telnet.TelnetConnection method*), 93
sort_list() (*robot.libraries.Collections.Collections method*), 53
Sortable (*class in robot.utils.sortable*), 296
source (*robot.model.testsuite.TestSuite attribute*), 187
source (*robot.parsing.model.InitFileSettingTable attribute*), 210
source (*robot.parsing.model.KeywordTable attribute*), 211
source (*robot.parsing.model.ResourceFileSettingTable attribute*), 210
source (*robot.parsing.model.TestCase attribute*), 211
source (*robot.parsing.model.TestCaseFileSettingTable attribute*), 209
source (*robot.parsing.model.TestCaseTable attribute*), 211
source (*robot.parsing.model.UserKeyword attribute*), 212
source (*robot.parsing.model.VariableTable attribute*), 210
source (*robot.parsing.settings.Arguments attribute*), 216
source (*robot.parsing.settings.Documentation attribute*), 215
source (*robot.parsing.settings.Fixture attribute*), 215
source (*robot.parsing.settings.Library attribute*), 217
source (*robot.parsing.settings.Metadata attribute*), 217
source (*robot.parsing.settings.Resource attribute*), 217
source (*robot.parsing.settings.Return attribute*), 216
source (*robot.parsing.settings.Setting attribute*), 214
source (*robot.parsing.settings.Tags attribute*), 216
source (*robot.parsing.settings.Template attribute*), 215
source (*robot.parsing.settings.Timeout attribute*), 216
source (*robot.parsing.settings.Variables attribute*), 218
source (*robot.result.executionresult.Result attribute*), 229
source (*robot.result.model.TestSuite attribute*), 247
source (*robot.running.model.TestSuite attribute*), 272
spacecount() (*robot.tidy.ArgumentValidator method*), 318
SpaceSeparatedTxtWriter (*class in robot.writer.filewriters*), 305
SpecDocBuilder (*class in robot.libdocpkg.specbuilder*), 23
split() (*robot.writer.rowsplitter.RowSplitter method*), 306
split_args_from_name_or_path() (*in module robot.utils.text*), 296
split_command_line() (*robot.libraries.Process.Process method*), 74
split_extension() (*robot.libraries.OperatingSystem.OperatingSystem method*), 67
split_from_equals() (*in module robot.utils.escaping*), 287
split_log (*robot.conf.settings.RebotSettings attribute*), 18
split_log (*robot.conf.settings.RobotSettings attribute*), 18
split_path() (*robot.libraries.OperatingSystem.OperatingSystem method*), 67
split_row() (*robot.parsing.tsvreader.TsvReader class method*), 220
split_string() (*robot.libraries.String.String method*), 82
split_string_from_right() (*robot.libraries.String.String method*), 82
split_string_to_characters() (*robot.libraries.String.String method*), 82
split_tags_from_doc() (*in module robot.utils.text*), 296
split_to_lines() (*robot.libraries.String.String method*), 79
SplitLogWriter (*class in robot.reporting.jswriter*), 222
start() (*robot.result.xmlelementhandlers.ArgumentHandler method*), 256
start() (*robot.result.xmlelementhandlers.ArgumentsHandler method*), 256
start() (*robot.result.xmlelementhandlers.AssignHandler method*), 255
start() (*robot.result.xmlelementhandlers.AssignVarHandler method*), 255
start() (*robot.result.xmlelementhandlers.DocHandler method*), 254
start() (*robot.result.xmlelementhandlers.ErrorsHandler method*), 256
start() (*robot.result.xmlelementhandlers.KeywordHandler method*), 253
start() (*robot.result.xmlelementhandlers.KeywordStatusHandler method*), 254
start() (*robot.result.xmlelementhandlers.MessageHandler method*), 254
start() (*robot.result.xmlelementhandlers.MetadataHandler method*), 254
start() (*robot.result.xmlelementhandlers.MetadataItemHandler method*), 255
start() (*robot.result.xmlelementhandlers.RobotHandler method*), 253
start() (*robot.result.xmlelementhandlers.RootHandler method*), 253
start() (*robot.result.xmlelementhandlers.RootSuiteHandler method*), 253
start() (*robot.result.xmlelementhandlers.StatisticsHandler method*), 256
start() (*robot.result.xmlelementhandlers.SuiteHandler*

method), 253
start() (*robot.result.xmlelementhandlers.SuiteStatusHandler method*), 254
start() (*robot.result.xmlelementhandlers.TagHandler method*), 255
start() (*robot.result.xmlelementhandlers.TagsHandler method*), 255
start() (*robot.result.xmlelementhandlers.TestCaseHandler method*), 253
start() (*robot.result.xmlelementhandlers.TestStatusHandler method*), 254
start() (*robot.result.xmlelementhandlers.TimeoutHandler method*), 255
start() (*robot.result.xmlelementhandlers.XmlElementHandler method*), 253
start() (*robot.running.timeouts.KeywordTimeout method*), 260
start() (*robot.running.timeouts.TestTimeout method*), 260
start() (*robot.utils.markupwriters.HtmlWriter method*), 290
start() (*robot.utils.markupwriters.NullMarkupWriter method*), 291
start() (*robot.utils.markupwriters.XmlWriter method*), 291
start_errors() (*robot.output.xmllogger.XmlLogger method*), 203
start_errors() (*robot.reporting.outputwriter.OutputWriter method*), 223
start_errors() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 226
start_errors() (*robot.result.visitor.ResultVisitor method*), 252
start_keyword() (*robot.conf.gatherfailed.GatherFailedSuites method*), 16
start_keyword() (*robot.conf.gatherfailed.GatherFailedTests method*), 15
start_keyword() (*robot.model.configurer.SuiteConfigurer method*), 170
start_keyword() (*robot.model.filter.EmptySuiteRemover method*), 171
start_keyword() (*robot.model.filter.Filter method*), 172
start_keyword() (*robot.model.modifier.ModelModifier method*), 178
start_keyword() (*robot.model.statistics.StatisticsBuilder method*), 180
start_keyword() (*robot.model.tagsetter.TagSetter method*), 184
start_keyword() (*robot.model.totalstatistics.TotalStatisticsBuilder method*), 190
start_keyword() (*robot.model.visitor.SuiteVisitor method*), 192
start_keyword() (*robot.output.console.dotted.StatusReporter method*), 193
start_keyword() (*robot.output.console.verbose.VerboseOutput method*), 195
start_keyword() (*robot.output.filelogger.FileLogger method*), 196
start_keyword() (*robot.output.logger.Logger method*), 199
start_keyword() (*robot.output.output.Output method*), 201
start_keyword() (*robot.output.xmllogger.XmlLogger method*), 203
start_keyword() (*robot.reporting.outputwriter.OutputWriter method*), 223
start_keyword() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 226
start_keyword() (*robot.result.configurer.SuiteConfigurer method*), 228
start_keyword() (*robot.result.keywordremover.AllKeywordsRemover method*), 232
start_keyword() (*robot.result.keywordremover.ByNameKeywordRemover method*), 234
start_keyword() (*robot.result.keywordremover.ByTagKeywordRemover method*), 235
start_keyword() (*robot.result.keywordremover.ForLoopItemsRemover method*), 236
start_keyword() (*robot.result.keywordremover.PassedKeywordRemover method*), 233
start_keyword() (*robot.result.keywordremover.WaitUntilKeywordSucceeds method*), 236
start_keyword() (*robot.result.keywordremover.WarningAndErrorFinder method*), 237
start_keyword() (*robot.result.merger.Merger method*), 239
start_keyword() (*robot.result.messagefilter.MessageFilter method*), 239
start_keyword() (*robot.result.resultbuilder.RemoveKeywords method*), 248
start_keyword() (*robot.result.suiteteardownfailed.SuiteTeardownFailed method*), 250
start_keyword() (*robot.result.suiteteardownfailed.SuiteTeardownFailureHandler method*), 249
start_keyword() (*robot.result.visitor.ResultVisitor method*), 252
start_keyword() (*robot.running.randomizer.Randomizer method*), 274
start_keyword() (*robot.running.runner.Runner method*), 275
start_keyword() (*robot.variables.scopes.SetVariables method*), 301
start_keyword() (*robot.variables.scopes.VariableScopes method*), 300
start_loggers (*robot.output.logger.Logger attribute*), 198
start_message() (*robot.conf.gatherfailed.GatherFailedSuites*

method), 16
start_message() (*robot.conf.gatherfailed.GatherFailedTests method*), 15
start_message() (*robot.model.configurer.SuiteConfigurer method*), 170
start_message() (*robot.model.filter.EmptySuiteRemover method*), 171
start_message() (*robot.model.filter.Filter method*), 172
start_message() (*robot.model.modifier.ModelModifier method*), 178
start_message() (*robot.model.statistics.StatisticsBuilder method*), 180
start_message() (*robot.model.tagsetter.TagSetter method*), 184
start_message() (*robot.model.totalstatistics.TotalStatisticsBuilder method*), 190
start_message() (*robot.model.visitor.SuiteVisitor method*), 192
start_message() (*robot.output.console.dotted.StatusReporter method*), 193
start_message() (*robot.output.xmllogger.XmlLogger method*), 204
start_message() (*robot.reporting.outputwriter.OutputWriter method*), 222
start_message() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 226
start_message() (*robot.result.configurer.SuiteConfigurer method*), 228
start_message() (*robot.result.keywordremover.AllKeywordsRemover method*), 232
start_message() (*robot.result.keywordremover.ByNameKeywordRemover method*), 234
start_message() (*robot.result.keywordremover.ByTagKeywordRemover method*), 235
start_message() (*robot.result.keywordremover.ForLoopItemsRemover method*), 236
start_message() (*robot.result.keywordremover.PassedKeywordRemover method*), 233
start_message() (*robot.result.keywordremover.WaitUntilKeywordSucceedsRemover method*), 237
start_message() (*robot.result.keywordremover.WarningAndErrorFinder method*), 238
start_message() (*robot.result.merger.Merger method*), 239
start_message() (*robot.result.messagefilter.MessageFilter method*), 240
start_message() (*robot.result.resultbuilder.RemoveKeywords method*), 248
start_message() (*robot.result.suiteteardownfailed.SuiteTeardownFailed method*), 250
start_message() (*robot.result.suiteteardownfailed.SuiteTeardownFailureHandler method*), 249
start_message() (*robot.result.visitor.ResultVisitor method*), 252
start_message() (*robot.running.randomizer.Randomizer method*), 274
start_message() (*robot.running.runner.Runner method*), 275
start_process() (*robot.libraries.Process.Process method*), 72
start_result() (*robot.output.xmllogger.XmlLogger method*), 204
start_result() (*robot.reporting.outputwriter.OutputWriter method*), 223
start_result() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 226
start_result() (*robot.result.visitor.ResultVisitor method*), 251
start_splitting_if_needed() (*robot.reporting.jsbuildingcontext.JsBuildingContext method*), 220
start_stat() (*robot.output.xmllogger.XmlLogger method*), 204
start_stat() (*robot.reporting.outputwriter.OutputWriter method*), 223
start_stat() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 226
start_stat() (*robot.result.visitor.ResultVisitor method*), 251
start_statistics() (*robot.output.xmllogger.XmlLogger method*), 203
start_statistics() (*robot.reporting.outputwriter.OutputWriter method*), 223
start_statistics() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 226
start_statistics() (*robot.result.visitor.ResultVisitor method*), 251
start_suite() (*robot.conf.gatherfailed.GatherFailedSuites method*), 15
start_suite() (*robot.conf.gatherfailed.GatherFailedTests method*), 15
start_suite() (*robot.model.configurer.SuiteConfigurer method*), 170
start_suite() (*robot.model.filter.EmptySuiteRemover method*), 171
start_suite() (*robot.model.filter.Filter method*), 172
start_suite() (*robot.model.modifier.ModelModifier method*), 179
start_suite() (*robot.model.statistics.StatisticsBuilder method*), 180
start_suite() (*robot.model.suitestatistics.SuiteStatisticsBuilder method*), 183

start_suite() (*robot.model.tagsetter.TagSetter method*), 184
start_suite() (*robot.model.totalstatistics.TotalStatisticsBuilder method*), 190
start_suite() (*robot.model.visitor.SuiteVisitor method*), 191
start_suite() (*robot.output.console.dotted.DottedOutput method*), 192
start_suite() (*robot.output.console.dotted.StatusReporter method*), 193
start_suite() (*robot.output.console.verbose.VerboseOutput method*), 195
start_suite() (*robot.output.filelogger.FileLogger method*), 196
start_suite() (*robot.output.logger.Logger method*), 199
start_suite() (*robot.output.output.Output method*), 201
start_suite() (*robot.output.xmllogger.XmlLogger method*), 203
start_suite() (*robot.reporting.outputwriter.OutputWriter method*), 223
start_suite() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 225
start_suite() (*robot.result.configurer.SuiteConfigurer method*), 228
start_suite() (*robot.result.keywordremover.AllKeywordsRemover method*), 232
start_suite() (*robot.result.keywordremover.ByNameKeywordRemover method*), 234
start_suite() (*robot.result.keywordremover.ByTagKeywordRemover method*), 235
start_suite() (*robot.result.keywordremover.ForLoopItemsRemover method*), 236
start_suite() (*robot.result.keywordremover.PassedKeywordRemover method*), 233
start_suite() (*robot.result.keywordremover.WaitUntilKeywordSucceedsRemover method*), 237
start_suite() (*robot.result.keywordremover.WarningAndErrorFinder method*), 237
start_suite() (*robot.result.merger.Merger method*), 238
start_suite() (*robot.result.messagefilter.MessageFilter method*), 240
start_suite() (*robot.result.resultbuilder.RemoveKeywords method*), 248
start_suite() (*robot.result.suiteteardownfailed.SuiteTeardownFailed method*), 250
start_suite() (*robot.result.suiteteardownfailed.SuiteTeardownFailureHandler method*), 249
start_suite() (*robot.result.visitor.ResultVisitor method*), 252
start_suite() (*robot.running.context.ExecutionContexts method*), 262
start_suite() (*robot.running.libraryscopes.GlobalScope method*), 264
start_suite() (*robot.running.libraryscopes.TestCaseScope method*), 265
start_suite() (*robot.running.libraryscopes.TestSuiteScope method*), 265
start_suite() (*robot.running.namespace.Namespace method*), 273
start_suite() (*robot.running.randomizer.Randomizer method*), 273
start_suite() (*robot.running.runner.Runner method*), 274
start_suite() (*robot.variables.scopes.SetVariables method*), 301
start_suite() (*robot.variables.scopes.VariableScopes method*), 300
start_suite_statistics() (*robot.output.xmllogger.XmlLogger method*), 203
start_suite_statistics() (*robot.reporting.outputwriter.OutputWriter method*), 223
start_suite_statistics() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 226
start_suite_statistics() (*robot.result.visitor.ResultVisitor method*), 251
start_table() (*robot.parsing.model.ResourceFile method*), 208
start_table() (*robot.parsing.model.TestCaseFile method*), 208
start_table() (*robot.parsing.model.TestDataDirectory method*), 209
start_table() (*robot.parsing.populators.FromFilePopulator method*), 213
start_test_statistics() (*robot.output.xmllogger.XmlLogger method*), 203
start_tag_statistics() (*robot.reporting.outputwriter.OutputWriter method*), 223
start_tag_statistics() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 226
start_tag_statistics() (*robot.result.visitor.ResultVisitor method*), 251
start_test() (*robot.conf.gatherfailed.GatherFailedSuites method*), 16
start_test() (*robot.conf.gatherfailed.GatherFailedTests method*), 15
start_test() (*robot.model.configurer.SuiteConfigurer method*), 170

start_test() (*robot.model.filter.EmptySuiteRemover method*), 172
start_test() (*robot.model.filter.Filter method*), 172
start_test() (*robot.model.modifier.ModelModifier method*), 179
start_test() (*robot.model.statistics.StatisticsBuilder method*), 180
start_test() (*robot.model.tagsetter.TagSetter method*), 184
start_test() (*robot.model.totalstatistics.TotalStatisticsBuilder method*), 190
start_test() (*robot.model.visitor.SuiteVisitor method*), 191
start_test() (*robot.output.console.dotted.StatusReporter method*), 193
start_test() (*robot.output.console.verbose.VerboseOutput method*), 195
start_test() (*robot.output.filelogger.FileLogger method*), 196
start_test() (*robot.output.logger.Logger method*), 199
start_test() (*robot.output.output.Output method*), 201
start_test() (*robot.output.xmllogger.XmlLogger method*), 203
start_test() (*robot.reporting.outputwriter.OutputWriter method*), 223
start_test() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 226
start_test() (*robot.result.configurer.SuiteConfigurer method*), 228
start_test() (*robot.result.keywordremover.AllKeywordsRemover method*), 232
start_test() (*robot.result.keywordremover.ByNameKeywordRemover method*), 234
start_test() (*robot.result.keywordremover.ByTagKeywordRemover method*), 235
start_test() (*robot.result.keywordremover.ForLoopItemsRemover method*), 236
start_test() (*robot.result.keywordremover.PassedKeywordRemover method*), 233
start_test() (*robot.result.keywordremover.WaitUntilKeywordSucceedsRemover method*), 237
start_test() (*robot.result.keywordremover.WarningAndErrorFinder method*), 237
start_test() (*robot.result.merger.Merger method*), 239
start_test() (*robot.result.messagefilter.MessageFilter method*), 240
start_test() (*robot.result.resultbuilder.RemoveKeywords method*), 248
start_test() (*robot.result.suiteteardownfailed.SuiteTeardownFailed method*), 250
start_test() (*robot.result.suiteteardownfailed.SuiteTeardownFailureHandler method*), 249
start_test() (*robot.result.visitor.ResultVisitor method*), 252
start_test() (*robot.running.libraryscopes.GlobalScope method*), 265
start_test() (*robot.running.libraryscopes.TestCaseScope method*), 265
start_test() (*robot.running.libraryscopes.TestSuiteScope method*), 265
start_test() (*robot.running.namespace.Namespace method*), 273
start_test() (*robot.running.randomizer.Randomizer method*), 274
start_test() (*robot.running.runner.Runner method*), 275
start_test() (*robot.variables.scopes.SetVariables method*), 301
start_test() (*robot.variables.scopes.VariableScopes method*), 300
start_total_statistics() (*robot.output.xmllogger.XmlLogger method*), 203
start_total_statistics() (*robot.reporting.outputwriter.OutputWriter method*), 223
start_total_statistics() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 226
start_total_statistics() (*robot.result.visitor.ResultVisitor method*), 251
start_user_keyword() (*robot.running.namespace.Namespace method*), 273
StartKeywordArguments (*class in robot.output.listenerarguments*), 197
starts_for_loop() (*robot.parsing.datarow.DataRow method*), 206
starts_test_or_user_keyword_setting() (*robot.parsing.datarow.DataRow method*), 206
StartSuiteArguments (*class in robot.output.listenerarguments*), 197
StartTestArguments (*class in robot.output.listenerarguments*), 197
starttime (*robot.result.model.Keyword attribute*), 242
starttime (*robot.result.model.TestCase attribute*), 243
starttime (*robot.result.model.TestSuite attribute*), 244
Stat (*class in robot.model.stats*), 181
stat (*robot.model.suitestatistics.SuiteStatistics attribute*), 183
stat_message (*robot.result.model.TestSuite attribute*), 245
stat_name() (*robot.libraries.dialogs_py.InputDialog*

method), 125
state() (*robot.libraries.dialogs_py.MessageDialog method*), 112
state() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 151
state() (*robot.libraries.dialogs_py.PassFailDialog method*), 164
state() (*robot.libraries.dialogs_py.SelectionDialog method*), 138
Statistics (*class in robot.model.statistics*), 179
statistics (*robot.result.executionresult.CombinedResult attribute*), 231
statistics (*robot.result.executionresult.Result attribute*), 229
statistics (*robot.result.model.TestSuite attribute*), 245
statistics_config (*robot.conf.settings.RebotSettings attribute*), 18
statistics_config (*robot.conf.settings.RobotSettings attribute*), 18
StatisticsBuilder (*class in robot.model.statistics*), 180
StatisticsBuilder (*class in robot.reporting.jsmodelbuilders*), 221
StatisticsHandler (*class in robot.result.xmlelementhandlers*), 256
status (*robot.errors.ContinueForLoop attribute*), 310
status (*robot.errors.ExecutionFailed attribute*), 308
status (*robot.errors.ExecutionFailures attribute*), 308
status (*robot.errors.ExecutionPassed attribute*), 309
status (*robot.errors.ExecutionStatus attribute*), 308
status (*robot.errors.ExitForLoop attribute*), 310
status (*robot.errors.HandlerExecutionFailed attribute*), 308
status (*robot.errors.PassExecution attribute*), 309
status (*robot.errors.ReturnFromKeyword attribute*), 310
status (*robot.errors.UserKeywordExecutionFailed attribute*), 309
status (*robot.result.model.Keyword attribute*), 241
status (*robot.result.model.TestCase attribute*), 243
status (*robot.result.model.TestSuite attribute*), 245
status (*robot.running.status.SuiteStatus attribute*), 276
status (*robot.running.status.TestStatus attribute*), 276
status() (*robot.output.console.verbose.VerboseWriter method*), 195
status_rc (*robot.conf.settings.RebotSettings attribute*), 18
status_rc (*robot.conf.settings.RobotSettings attribute*), 18
StatusReporter (*class in robot.output.console.dotted*), 193
StatusReporter (*class in robot.running.statusreporter*), 277
stderr (*robot.libraries.Process.ExecutionResult attribute*), 74
stdout (*robot.libraries.Process.ExecutionResult attribute*), 74
StdoutLogSplitter (*class in robot.output.stdoutlogsplitter*), 203
Step (*class in robot.parsing.model*), 212
StepBuilder (*class in robot.running.builder*), 262
StepPopulator (*class in robot.parsing.tablepopulators*), 219
StepRunner (*class in robot.running.steprunner*), 277
StoredFinder (*class in robot.variables.finders*), 298
String (*class in robot.libraries.String*), 78
string() (*robot.reporting.jsbuildingcontext.JsBuildingContext method*), 220
string_value() (*robot.parsing.settings.StringValueJoiner method*), 214
StringCache (*class in robot.reporting.stringcache*), 225
StringDumper (*class in robot.htmldata.jsonwriter*), 20
StringIndex (*class in robot.reporting.stringcache*), 224
strings (*robot.reporting.jsbuildingcontext.JsBuildingContext attribute*), 220
StringValueJoiner (*class in robot.parsing.settings*), 214
strip() (*robot.libraries.XML.NameSpaceStripper method*), 104
strip_string() (*robot.libraries.String.String method*), 82
subtract_date_from_date() (*in module robot.libraries.DateTime*), 57
subtract_time_from_date() (*in module robot.libraries.DateTime*), 57
subtract_time_from_time() (*in module robot.libraries.DateTime*), 58
suite (*robot.model.statistics.Statistics attribute*), 179
suite (*robot.result.executionresult.Result attribute*), 229
suite_config (*robot.conf.settings.RebotSettings attribute*), 18
suite_config (*robot.conf.settings.RobotSettings attribute*), 17
suite_separator() (*robot.output.console.verbose.VerboseWriter method*), 195
suite_teardown_failed() (*robot.result.model.TestSuite method*), 247
SuiteBuilder (*class in robot.reporting.jsmodelbuilders*), 221
SuiteConfigurer (*class in robot.model.configurer*), 170
SuiteConfigurer (*class in robot.result.configurer*), 228

SuiteHandler (*class in robot.result.xmlelementhandlers*), 253
SuiteMessage (*class in robot.running.status*), 277
SuiteNamePatterns (*class in robot.model.namepatterns*), 179
suites (*robot.model.suitestatistics.SuiteStatistics attribute*), 183
suites (*robot.model.testsuite.TestSuite attribute*), 187
suites (*robot.result.model.TestSuite attribute*), 247
suites (*robot.running.model.TestSuite attribute*), 272
SuiteStat (*class in robot.model.stats*), 181
SuiteStatistics (*class in robot.model.suitestatistics*), 183
SuiteStatisticsBuilder (*class in robot.model.suitestatistics*), 183
SuiteStatus (*class in robot.running.status*), 276
SuiteStatusHandler (*class in robot.result.xmlelementhandlers*), 254
SuiteTeardownFailed (*class in robot.result.suiteteardownfailed*), 250
SuiteTeardownFailureHandler (*class in robot.result.suiteteardownfailed*), 249
SuiteVisitor (*class in robot.model.visitor*), 191
SuiteWriter (*class in robot.reporting.jswriter*), 222
supports_kwargs (*robot.running.dynamicmethods.RunKeyword attribute*), 262
switch() (*robot.utils.connectioncache.ConnectionCache method*), 284
switch_connection() (*robot.libraries.Telnet.Telnet method*), 87
switch_process() (*robot.libraries.Process.Process method*), 74
system_decode() (*in module robot.utils.encoding*), 286
system_encode() (*in module robot.utils.encoding*), 286

T

table_end() (*robot.parsing.htmlreader.HtmlReader method*), 206
table_start() (*robot.parsing.htmlreader.HtmlReader method*), 206
TableFormatter (*class in robot.utils.htmlformatters*), 288
tag (*robot.result.xmlelementhandlers.ArgumentHandler attribute*), 256
tag (*robot.result.xmlelementhandlers.ArgumentsHandler attribute*), 256
tag (*robot.result.xmlelementhandlers.AssignHandler attribute*), 255
tag (*robot.result.xmlelementhandlers.AssignVarHandler attribute*), 255
tag (*robot.result.xmlelementhandlers.DocHandler attribute*), 254
tag (*robot.result.xmlelementhandlers.ErrorsHandler attribute*), 256
tag (*robot.result.xmlelementhandlers.KeywordHandler attribute*), 253
tag (*robot.result.xmlelementhandlers.KeywordStatusHandler attribute*), 254
tag (*robot.result.xmlelementhandlers.MessageHandler attribute*), 254
tag (*robot.result.xmlelementhandlers.MetadataHandler attribute*), 254
tag (*robot.result.xmlelementhandlers.MetadataItemHandler attribute*), 255
tag (*robot.result.xmlelementhandlers.RobotHandler attribute*), 253
tag (*robot.result.xmlelementhandlers.RootSuiteHandler attribute*), 253
tag (*robot.result.xmlelementhandlers.StatisticsHandler attribute*), 256
tag (*robot.result.xmlelementhandlers.SuiteHandler attribute*), 253
tag (*robot.result.xmlelementhandlers.SuiteStatusHandler attribute*), 254
tag (*robot.result.xmlelementhandlers.TagHandler attribute*), 255
tag (*robot.result.xmlelementhandlers.TagsHandler attribute*), 255
tag (*robot.result.xmlelementhandlers.TestCaseHandler attribute*), 253
tag (*robot.result.xmlelementhandlers.TestStatusHandler attribute*), 254
tag (*robot.result.xmlelementhandlers.TimeoutHandler attribute*), 255
tag_is_critical() (*robot.model.criticality.Criticality method*), 171
tag_is_non_critical() (*robot.model.criticality.Criticality method*), 171
TagHandler (*class in robot.result.xmlelementhandlers*), 255
TagPattern() (*in module robot.model.tags*), 183
TagPatterns (*class in robot.model.tags*), 183
Tags (*class in robot.model.tags*), 183
Tags (*class in robot.parsing.settings*), 216
tags (*robot.model.keyword.Keyword attribute*), 175
tags (*robot.model.statistics.Statistics attribute*), 179
tags (*robot.model.tagstatistics.TagStatistics attribute*), 185
tags (*robot.model.testcase.TestCase attribute*), 186
tags (*robot.result.model.Keyword attribute*), 243
tags (*robot.result.model.TestCase attribute*), 244
tags (*robot.running.model.ForLoop attribute*), 268
tags (*robot.running.model.Keyword attribute*), 266
tags (*robot.running.model.TestCase attribute*), 269

tags (*robot.running.model.UserKeyword attribute*), 272
TagSetter (*class in robot.model.tagsetter*), 184
TagsHandler (*class in robot.result.xmlelementhandlers*), 255
TagStat (*class in robot.model.stats*), 182
TagStatDoc (*class in robot.model.tagstatistics*), 185
TagStatInfo (*class in robot.model.tagstatistics*), 185
TagStatistics (*class in robot.model.tagstatistics*), 185
TagStatisticsBuilder (*class in robot.model.tagstatistics*), 185
TagStatLink (*class in robot.model.tagstatistics*), 185
tail (*robot.parsing.datarow.DataRow attribute*), 205
take_screenshot() (*robot.libraries.Screenshot.Screenshot method*), 78
take_screenshot_without_embedding() (*robot.libraries.Screenshot.Screenshot method*), 78
td_end() (*robot.parsing.htmlreader.HtmlReader method*), 206
td_start() (*robot.parsing.htmlreader.HtmlReader method*), 206
teardown (*robot.model.keyword.Keywords attribute*), 176
teardown_allowed (*robot.running.status.Exit attribute*), 276
teardown_allowed (*robot.running.status.SuiteStatus attribute*), 276
teardown_allowed (*robot.running.status.TestStatus attribute*), 276
teardown_executed() (*robot.running.status.SuiteStatus method*), 276
teardown_executed() (*robot.running.status.TestStatus method*), 276
teardown_message (*robot.running.status.ParentMessage attribute*), 277
teardown_message (*robot.running.status.SuiteMessage attribute*), 277
teardown_message (*robot.running.status.TestMessage attribute*), 276
TEARDOWN_TYPE (*robot.model.keyword.Keyword attribute*), 174
TEARDOWN_TYPE (*robot.result.model.Keyword attribute*), 242
TEARDOWN_TYPE (*robot.running.model.ForLoop attribute*), 267
TEARDOWN_TYPE (*robot.running.model.Keyword attribute*), 266
Telnet (*class in robot.libraries.Telnet*), 84
TelnetConnection (*class in robot.libraries.Telnet*), 88
Template (*class in robot.parsing.settings*), 215
template (*robot.running.model.TestCase attribute*), 268
TerminalEmulator (*class in robot.libraries.Telnet*), 93
terminate_all_processes() (*robot.libraries.Process.Process method*), 73
terminate_process() (*robot.libraries.Process.Process method*), 73
TERMINATE_TIMEOUT (*robot.libraries.Process.Process attribute*), 72
test() (*robot.libraries.Screenshot.ScreenshotTaker method*), 78
TEST_CASE_FILE_TYPE (*robot.running.handlerstore.HandlerStore attribute*), 263
TEST_CASE_FILE_TYPE (*robot.running.userkeyword.UserLibrary attribute*), 278
test_class (*robot.model.testsuite.TestSuite attribute*), 187
test_class (*robot.result.model.TestSuite attribute*), 244
test_class (*robot.running.model.TestSuite attribute*), 269
test_count (*robot.model.testsuite.TestSuite attribute*), 188
test_count (*robot.result.model.TestSuite attribute*), 247
test_count (*robot.running.model.TestSuite attribute*), 272
test_failed() (*robot.running.status.TestStatus method*), 276
test_is_critical() (*robot.model.criticality.Criticality method*), 171
TEST_LIBRARY_TYPE (*robot.running.handlerstore.HandlerStore attribute*), 263
test_or_user_keyword_setting_name() (*robot.parsing.datarow.DataRow method*), 206
test_separator() (*robot.output.console.verbose.VerboseWriter method*), 195
TestBuilder (*class in robot.reporting.jsmodelbuilders*), 221
TestCase (*class in robot.model.testcase*), 186
TestCase (*class in robot.parsing.model*), 211
TestCase (*class in robot.result.model*), 243
TestCase (*class in robot.running.model*), 268
TestCaseFile (*class in robot.parsing.model*), 207
TestCaseFileSettingTable (*class in*

robot.parsing.model), 209
TestCaseHandler (*class in robot.result.xmlelementhandlers*), 253
TestCasePopulator (*class in robot.parsing.tablepopulators*), 219
TestCases (*class in robot.model.testcase*), 187
TestCaseScope (*class in robot.running.libraryscopes*), 265
TestCaseTable (*class in robot.parsing.model*), 210
TestData() (*in module robot.parsing.model*), 207
TestDataDirectory (*class in robot.parsing.model*), 208
TestDefaults (*class in robot.running.defaults*), 262
TestDoc (*class in robot.testdoc*), 316
testdoc() (*in module robot.testdoc*), 317
testdoc_cli() (*in module robot.testdoc*), 316
TestdocModelWriter (*class in robot.testdoc*), 316
TestLibrary() (*in module robot.running.testlibraries*), 278
TestMessage (*class in robot.running.status*), 276
TestNamePatterns (*class in robot.model.namepatterns*), 179
tests (*robot.model.testsuite.TestSuite attribute*), 187
tests (*robot.result.model.TestSuite attribute*), 247
tests (*robot.running.model.TestSuite attribute*), 272
TestStatus (*class in robot.running.status*), 276
TestStatusHandler (*class in robot.result.xmlelementhandlers*), 254
TestSuite (*class in robot.model.testsuite*), 187
TestSuite (*class in robot.result.model*), 244
TestSuite (*class in robot.running.model*), 269
TestSuiteBuilder (*class in robot.running.builder*), 261
TestSuites (*class in robot.model.testsuite*), 189
TestSuiteScope (*class in robot.running.libraryscopes*), 265
TestTablePopulator (*class in robot.parsing.tablepopulators*), 218
TestTimeout (*class in robot.running.timeouts*), 260
TestValues (*class in robot.running.defaults*), 262
Tidy (*class in robot.tidy*), 317
tidy_cli() (*in module robot.tidy*), 318
TidyCommandLine (*class in robot.tidy*), 318
time_left() (*robot.running.timeouts.KeywordTimeout method*), 260
time_left() (*robot.running.timeouts.TestTimeout method*), 260
timed_out() (*robot.running.timeouts.KeywordTimeout method*), 260
timed_out() (*robot.running.timeouts.TestTimeout method*), 260
Timeout (*class in robot.parsing.settings*), 215
Timeout (*class in robot.running.model*), 272
Timeout (*class in robot.running.timeouts.posix*), 261
Timeout (*class in robot.running.timeouts.windows*), 261
timeout (*robot.errors.ContinueForLoop attribute*), 310
timeout (*robot.errors.ExecutionFailed attribute*), 308
timeout (*robot.errors.ExecutionFailures attribute*), 308
timeout (*robot.errors.ExecutionPassed attribute*), 309
timeout (*robot.errors.ExecutionStatus attribute*), 307
timeout (*robot.errors.ExitForLoop attribute*), 310
timeout (*robot.errors.HandlerExecutionFailed attribute*), 308
timeout (*robot.errors.PassExecution attribute*), 309
timeout (*robot.errors.ReturnFromKeyword attribute*), 311
timeout (*robot.errors.UserKeywordExecutionFailed attribute*), 309
timeout (*robot.model.keyword.Keyword attribute*), 174
timeout (*robot.model.testcase.TestCase attribute*), 186
timeout (*robot.result.model.Keyword attribute*), 243
timeout (*robot.result.model.TestCase attribute*), 244
timeout (*robot.running.model.ForLoop attribute*), 268
timeout (*robot.running.model.Keyword attribute*), 266
timeout (*robot.running.model.TestCase attribute*), 268
timeout (*robot.running.model.UserKeyword attribute*), 272
TimeoutError, 307
TimeoutHandler (*class in robot.result.xmlelementhandlers*), 255
TimeoutHTTPSTransport (*class in robot.libraries.Remote*), 76
TimeoutHTTPTransport (*class in robot.libraries.Remote*), 76
timestamp (*robot.model.message.Message attribute*), 176
timestamp (*robot.output.loggerhelper.Message attribute*), 200
timestamp (*robot.result.model.Message attribute*), 241
timestamp() (*robot.reporting.jsbuildingcontext.JsBuildingContext method*), 220
timestamp_to_secs() (*in module robot.utils.robottime*), 295
TimestampCache (*class in robot.utils.robottime*), 295
timestr_to_secs() (*in module robot.utils.robottime*), 294
title() (*robot.libraries.dialogs_py.InputDialog method*), 125
title() (*robot.libraries.dialogs_py.MessageDialog method*), 112
title() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 151
title() (*robot.libraries.dialogs_py.PassFailDialog method*), 164
title() (*robot.libraries.dialogs_py.SelectionDialog method*), 138
tk_bisque() (*robot.libraries.dialogs_py.InputDialog*

method), 125
tk_bisque() (*robot.libraries.dialogs_py.MessageDialog method*), 112
tk_bisque() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 151
tk_bisque() (*robot.libraries.dialogs_py.PassFailDialog method*), 164
tk_bisque() (*robot.libraries.dialogs_py.SelectionDialog method*), 138
tk_focusFollowsMouse() (*robot.libraries.dialogs_py.InputDialog method*), 125
tk_focusFollowsMouse() (*robot.libraries.dialogs_py.MessageDialog method*), 112
tk_focusFollowsMouse() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 151
tk_focusFollowsMouse() (*robot.libraries.dialogs_py.PassFailDialog method*), 164
tk_focusFollowsMouse() (*robot.libraries.dialogs_py.SelectionDialog method*), 138
tk_focusNext() (*robot.libraries.dialogs_py.InputDialog method*), 125
tk_focusNext() (*robot.libraries.dialogs_py.MessageDialog method*), 112
tk_focusNext() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 151
tk_focusNext() (*robot.libraries.dialogs_py.PassFailDialog method*), 164
tk_focusNext() (*robot.libraries.dialogs_py.SelectionDialog method*), 138
tk_focusPrev() (*robot.libraries.dialogs_py.InputDialog method*), 125
tk_focusPrev() (*robot.libraries.dialogs_py.MessageDialog method*), 112
tk_focusPrev() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 151
tk_focusPrev() (*robot.libraries.dialogs_py.PassFailDialog method*), 164
tk_focusPrev() (*robot.libraries.dialogs_py.SelectionDialog method*), 138
tk_menuBar() (*robot.libraries.dialogs_py.InputDialog method*), 125
tk_menuBar() (*robot.libraries.dialogs_py.MessageDialog method*), 112
tk_menuBar() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 151
tk_menuBar() (*robot.libraries.dialogs_py.PassFailDialog method*), 164
tk_menuBar() (*robot.libraries.dialogs_py.SelectionDialog method*), 138
tk_setPalette() (*robot.libraries.dialogs_py.InputDialog method*), 125
tk_setPalette() (*robot.libraries.dialogs_py.MessageDialog method*), 112
tk_setPalette() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 151
tk_setPalette() (*robot.libraries.dialogs_py.PassFailDialog method*), 164
tk_setPalette() (*robot.libraries.dialogs_py.SelectionDialog method*), 138
tk_strictMotif() (*robot.libraries.dialogs_py.InputDialog method*), 125
tk_strictMotif() (*robot.libraries.dialogs_py.MessageDialog method*), 112
tk_strictMotif() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 151
tk_strictMotif() (*robot.libraries.dialogs_py.PassFailDialog method*), 164
tk_strictMotif() (*robot.libraries.dialogs_py.SelectionDialog method*), 138
tkraise() (*robot.libraries.dialogs_py.InputDialog method*), 125
tkraise() (*robot.libraries.dialogs_py.MessageDialog method*), 112
tkraise() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 151
tkraise() (*robot.libraries.dialogs_py.PassFailDialog method*), 164
tkraise() (*robot.libraries.dialogs_py.SelectionDialog method*), 138
to_log (*robot.running.context.ExecutionContexts attribute*), 262
total (*robot.model.statistics.Statistics attribute*), 179
total (*robot.model.stats.CombinedTagStat attribute*), 182
total (*robot.model.stats.CriticalTagStat attribute*), 183
total (*robot.model.stats.Stat attribute*), 181
total (*robot.model.stats.SuiteStat attribute*), 181
total (*robot.model.stats.TagStat attribute*), 182
total (*robot.model.stats.TotalStat attribute*), 181
TotalStat (*class in robot.model.stats*), 181
TotalStatistics (*class in robot.model.totalstatistics*), 189
TotalStatisticsBuilder (*class in robot.model.totalstatistics*), 189
touch() (*robot.libraries.OperatingSystem.OperatingSystem method*), 69
tr_end() (*robot.parsing.htmlreader.HtmlReader method*), 206
tr_start() (*robot.parsing.htmlreader.HtmlReader method*), 206
trace() (*in module robot.api.logger*), 14
trace() (*in module robot.output.librarylogger*), 196
trace() (*robot.output.filelogger.FileLogger method*),

196
trace() (*robot.output.logger.Logger method*), 199
trace() (*robot.output.loggerhelper.AbstractLogger method*), 200
trace() (*robot.output.output.Output method*), 201
traceback (*robot.utils.error.JavaErrorDetails attribute*), 287
traceback (*robot.utils.error.PythonErrorDetails attribute*), 287
transient() (*robot.libraries.dialogs_py.InputDialog method*), 125
transient() (*robot.libraries.dialogs_py.MessageDialog method*), 112
transient() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 151
transient() (*robot.libraries.dialogs_py.PassFailDialog method*), 164
transient() (*robot.libraries.dialogs_py.SelectionDialog method*), 138
tsv_column_count (*robot.writer.datafilewriter.WritingContext attribute*), 304
tsv_format (*robot.writer.datafilewriter.WritingContext attribute*), 304
TsvFileWriter (*class in robot.writer.filewriters*), 305
TsvFormatter (*class in robot.writer.formatters*), 305
TsvReader (*class in robot.parsing.tsvreader*), 220
TupleListDumper (*class in robot.htmldata.jsonwriter*), 20
txt_column_count (*robot.writer.datafilewriter.WritingContext attribute*), 304
txt_format (*robot.writer.datafilewriter.WritingContext attribute*), 304
TxtFormatter (*class in robot.writer.formatters*), 305
type (*robot.model.keyword.Keyword attribute*), 174
type (*robot.model.stats.CombinedTagStat attribute*), 182
type (*robot.model.stats.CriticalTagStat attribute*), 183
type (*robot.model.stats.SuiteStat attribute*), 181
type (*robot.model.stats.TagStat attribute*), 182
type (*robot.model.stats.TotalStat attribute*), 181
type (*robot.parsing.model.InitFileSettingTable attribute*), 210
type (*robot.parsing.model.KeywordTable attribute*), 211
type (*robot.parsing.model.ResourceFileSettingTable attribute*), 210
type (*robot.parsing.model.TestCaseFileSettingTable attribute*), 209
type (*robot.parsing.model.TestCaseTable attribute*), 210
type (*robot.parsing.model.VariableTable attribute*), 210
type (*robot.parsing.settings.Library attribute*), 217
type (*robot.parsing.settings.Resource attribute*), 217
type (*robot.parsing.settings.Variables attribute*), 218
type (*robot.result.model.Keyword attribute*), 243
type (*robot.running.model.ForLoop attribute*), 268
type (*robot.running.model.Keyword attribute*), 266
type (*robot.running.timeouts.KeywordTimeout attribute*), 261
type (*robot.running.timeouts.TestTimeout attribute*), 260
type_name() (*in module robot.utils.robottypes2*), 296
types (*robot.running.arguments.argumentspec.ArgumentSpec attribute*), 259

U

unbind() (*robot.libraries.dialogs_py.InputDialog method*), 125
unbind() (*robot.libraries.dialogs_py.MessageDialog method*), 112
unbind() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 151
unbind() (*robot.libraries.dialogs_py.PassFailDialog method*), 164
unbind() (*robot.libraries.dialogs_py.SelectionDialog method*), 138
unbind_all() (*robot.libraries.dialogs_py.InputDialog method*), 125
unbind_all() (*robot.libraries.dialogs_py.MessageDialog method*), 112
unbind_all() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 151
unbind_all() (*robot.libraries.dialogs_py.PassFailDialog method*), 164
unbind_all() (*robot.libraries.dialogs_py.SelectionDialog method*), 138
unbind_class() (*robot.libraries.dialogs_py.InputDialog method*), 125
unbind_class() (*robot.libraries.dialogs_py.MessageDialog method*), 112
unbind_class() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 151
unbind_class() (*robot.libraries.dialogs_py.PassFailDialog method*), 164
unbind_class() (*robot.libraries.dialogs_py.SelectionDialog method*), 138
unescape() (*in module robot.utils.escaping*), 287
unescape() (*robot.parsing.htmlreader.HtmlReader method*), 207
unescape() (*robot.utils.escaping.Unescaper method*), 287
Unescaper (*class in robot.utils.escaping*), 287
unic() (*in module robot.utils.unic*), 296
unknown_decl() (*robot.parsing.htmlreader.HtmlReader method*), 206
unregister() (*robot.output.listenermethods.LibraryListenerMethods method*), 197
unregister() (*robot.output.listeners.LibraryListeners method*), 198

unregister_console_logger() (*robot.output.logger.Logger method*), 198
unregister_logger() (*robot.output.logger.Logger method*), 199
unregister_xml_logger() (*robot.output.logger.Logger method*), 198
unstrip() (*robot.libraries.XML.NameSpaceStripper method*), 104
update() (*robot.libraries.dialogs_py.InputDialog method*), 126
update() (*robot.libraries.dialogs_py.MessageDialog method*), 113
update() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 152
update() (*robot.libraries.dialogs_py.PassFailDialog method*), 165
update() (*robot.libraries.dialogs_py.SelectionDialog method*), 139
update() (*robot.model.metadata.Metadata method*), 177
update() (*robot.utils.dotdict.DotDict method*), 286
update() (*robot.utils.normalizing.NormalizedDict method*), 292
update() (*robot.variables.scopes.GlobalVariables method*), 301
update() (*robot.variables.scopes.SetVariables method*), 301
update() (*robot.variables.store.VariableStore method*), 301
update() (*robot.variables.variables.Variables method*), 303
update_idletasks() (*robot.libraries.dialogs_py.InputDialog method*), 126
update_idletasks() (*robot.libraries.dialogs_py.MessageDialog method*), 113
update_idletasks() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 152
update_idletasks() (*robot.libraries.dialogs_py.PassFailDialog method*), 165
update_idletasks() (*robot.libraries.dialogs_py.SelectionDialog method*), 139
updatepos() (*robot.parsing.htmlreader.HtmlReader method*), 207
user_agent (*robot.libraries.Remote.TimeoutHTTPSTransport attribute*), 76
user_agent (*robot.libraries.Remote.TimeoutHTTPTransport attribute*), 76
UserErrorHandler (*class in robot.running.usererrorhandler*), 278
UserKeyword (*class in robot.parsing.model*), 211
UserKeyword (*class in robot.running.model*), 272
UserKeywordArgumentParser (*class in robot.running.arguments.argumentparser*), 258
UserKeywordExecutionFailed, 308
UserKeywordHandler (*class in robot.running.userkeyword*), 278
UserKeywordPopulator (*class in robot.parsing.tablepopulators*), 219
UserKeywordRunner (*class in robot.running.userkeywordrunner*), 279
UserLibrary (*class in robot.running.userkeyword*), 278
Utf8Reader (*class in robot.utils.utf8reader*), 297

V

validate() (*robot.libdoc.LibDoc method*), 311
validate() (*robot.rebot.Rebot method*), 313
validate() (*robot.run.RobotFramework method*), 314
validate() (*robot.running.arguments.argumentvalidator.ArgumentValid method*), 260
validate() (*robot.testdoc.TestDoc method*), 316
validate() (*robot.tidy.TidyCommandLine method*), 318
validate() (*robot.utils.application.Application method*), 280
validate() (*robot.variables.assigner.AssignmentValidator method*), 297
validate_assignment() (*robot.variables.assigner.VariableAssignment method*), 297
validate_command() (*robot.libdocpkg.consoleviewer.ConsoleViewer class method*), 21
validate_flatten_keyword() (*in module robot.result.flattenkeywordmatcher*), 232
validate_var() (*in module robot.variables.isvar*), 299
value (*robot.parsing.comments.Comments attribute*), 205
values (*robot.running.model.ForLoop attribute*), 267
values() (*robot.model.metadata.Metadata method*), 178
values() (*robot.running.importer.ImportCache method*), 264
values() (*robot.utils.dotdict.DotDict method*), 286
values() (*robot.utils.normalizing.NormalizedDict method*), 292
Variable (*class in robot.parsing.model*), 211
Variable (*class in robot.running.model*), 272
variable_files (*robot.conf.settings.RobotSettings attribute*), 17

variable_not_found() (*in module robot.variables.notfound*), 299
variable_should_exist() (*robot.libraries.BuiltIn.BuiltIn method*), 45
variable_should_not_exist() (*robot.libraries.BuiltIn.BuiltIn method*), 45
VariableAssigner (*class in robot.variables.assigner*), 297
VariableAssignment (*class in robot.variables.assigner*), 297
VariableError, 307
VariableFileSetter (*class in robot.variables.filesetter*), 298
VariableFinder (*class in robot.variables.finders*), 298
VariableIterator (*class in robot.variables.splitter*), 301
VariablePopulator (*class in robot.parsing.tablepopulators*), 219
VariableReplacer (*class in robot.running.arguments.argumentresolver*), 259
VariableReplacer (*class in robot.variables.replacer*), 299
Variables (*class in robot.parsing.settings*), 217
Variables (*class in robot.variables.variables*), 302
variables (*robot.conf.settings.RobotSettings attribute*), 17
variables (*robot.running.model.ForLoop attribute*), 267
variables (*robot.running.model.ResourceFile attribute*), 272
variables() (*robot.model.imports.Imports method*), 173
VariableScopes (*class in robot.variables.scopes*), 300
VariableSplitter (*class in robot.variables.splitter*), 301
VariableStore (*class in robot.variables.store*), 301
VariableTable (*class in robot.parsing.model*), 210
VariableTablePopulator (*class in robot.parsing.tablepopulators*), 218
VariableTableReader (*class in robot.variables.tablesetter*), 302
VariableTableSetter (*class in robot.variables.tablesetter*), 302
VariableTableValue() (*in module robot.variables.tablesetter*), 302
VariableTableValueBase (*class in robot.variables.tablesetter*), 302
VerboseOutput (*class in robot.output.console.verbose*), 195
VerboseWriter (*class in robot.output.console.verbose*), 195
version() (*robot.libdocpkg.consoleviewer.ConsoleViewer method*), 21
view() (*robot.libdocpkg.consoleviewer.ConsoleViewer method*), 21
viewitems() (*robot.utils.dotdict.DotDict method*), 286
viewkeys() (*robot.utils.dotdict.DotDict method*), 286
viewvalues() (*robot.utils.dotdict.DotDict method*), 286
visit() (*robot.model.imports.Imports method*), 173
visit() (*robot.model.itemlist.ItemList method*), 174
visit() (*robot.model.keyword.Keyword method*), 175
visit() (*robot.model.keyword.Keywords method*), 176
visit() (*robot.model.message.Message method*), 176
visit() (*robot.model.message.Messages method*), 177
visit() (*robot.model.statistics.Statistics method*), 180
visit() (*robot.model.stats.CombinedTagStat method*), 182
visit() (*robot.model.stats.CriticalTagStat method*), 183
visit() (*robot.model.stats.Stat method*), 181
visit() (*robot.model.stats.SuiteStat method*), 181
visit() (*robot.model.stats.TagStat method*), 182
visit() (*robot.model.stats.TotalStat method*), 181
visit() (*robot.model.suitestatistics.SuiteStatistics method*), 183
visit() (*robot.model.tagstatistics.TagStatistics method*), 185
visit() (*robot.model.testcase.TestCase method*), 186
visit() (*robot.model.testcase.TestCases method*), 187
visit() (*robot.model.testsuite.TestSuite method*), 189
visit() (*robot.model.testsuite.TestSuites method*), 189
visit() (*robot.model.totalstatistics.TotalStatistics method*), 189
visit() (*robot.output.loggerhelper.Message method*), 200
visit() (*robot.result.executionerrors.ExecutionErrors method*), 229
visit() (*robot.result.executionresult.CombinedResult method*), 231
visit() (*robot.result.executionresult.Result method*), 230
visit() (*robot.result.model.Keyword method*), 243
visit() (*robot.result.model.Message method*), 241
visit() (*robot.result.model.TestCase method*), 244
visit() (*robot.result.model.TestSuite method*), 247
visit() (*robot.running.model.ForLoop method*), 268
visit() (*robot.running.model.Keyword method*), 266
visit() (*robot.running.model.TestCase method*), 269
visit() (*robot.running.model.TestSuite method*), 272
visit_errors() (*robot.output.xmllogger.XmlLogger method*), 204

visit_errors() (*robot.reporting.outputwriter.OutputWriter method*), 223
visit_errors() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 225
visit_errors() (*robot.result.visitor.ResultVisitor method*), 252
visit_keyword() (*robot.conf.gatherfailed.GatherFailedSuites method*), 16
visit_keyword() (*robot.conf.gatherfailed.GatherFailedTests method*), 15
visit_keyword() (*robot.model.configurer.SuiteConfigurer method*), 170
visit_keyword() (*robot.model.filter.EmptySuiteRemover method*), 171
visit_keyword() (*robot.model.filter.Filter method*), 172
visit_keyword() (*robot.model.modifier.ModelModifier method*), 179
visit_keyword() (*robot.model.statistics.StatisticsBuilder method*), 180
visit_keyword() (*robot.model.tagsetter.TagSetter method*), 184
visit_keyword() (*robot.model.totalstatistics.TotalStatisticsBuilder method*), 189
visit_keyword() (*robot.model.visitor.SuiteVisitor method*), 192
visit_keyword() (*robot.output.console.dotted.StatusReporter method*), 193
visit_keyword() (*robot.output.xmllogger.XmlLogger method*), 204
visit_keyword() (*robot.reporting.outputwriter.OutputWriter method*), 222
visit_keyword() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 225
visit_keyword() (*robot.result.configurer.SuiteConfigurer method*), 229
visit_keyword() (*robot.result.keywordremover.AllKeywordsRemover method*), 232
visit_keyword() (*robot.result.keywordremover.ByNameKeywordRemover method*), 234
visit_keyword() (*robot.result.keywordremover.ByTagKeywordRemover method*), 235
visit_keyword() (*robot.result.keywordremover.ForLoopItemsRemover method*), 236
visit_keyword() (*robot.result.keywordremover.PassedKeywordRemover method*), 233
visit_keyword() (*robot.result.keywordremover.WaitUntilKeywordSucceedsRemover method*), 237
visit_keyword() (*robot.result.keywordremover.WarningAndErrorFinder method*), 238
visit_keyword() (*robot.result.merger.Merger method*), 239
visit_keyword() (*robot.result.messagefilter.MessageFilter method*), 240
visit_keyword() (*robot.result.resultbuilder.RemoveKeywords method*), 249
visit_keyword() (*robot.result.suiteteardownfailed.SuiteTeardownFailed method*), 250
visit_keyword() (*robot.result.suiteteardownfailed.SuiteTeardownFailure method*), 249
visit_keyword() (*robot.result.visitor.ResultVisitor method*), 252
visit_keyword() (*robot.running.randomizer.Randomizer method*), 274
visit_keyword() (*robot.running.runner.Runner method*), 275
visit_message() (*robot.conf.gatherfailed.GatherFailedSuites method*), 16
visit_message() (*robot.conf.gatherfailed.GatherFailedTests method*), 16
visit_message() (*robot.model.configurer.SuiteConfigurer method*), 170
visit_message() (*robot.model.filter.EmptySuiteRemover method*), 172
visit_message() (*robot.model.filter.Filter method*), 173
visit_message() (*robot.model.modifier.ModelModifier method*), 179
visit_message() (*robot.model.statistics.StatisticsBuilder method*), 180
visit_message() (*robot.model.tagsetter.TagSetter method*), 184
visit_message() (*robot.model.totalstatistics.TotalStatisticsBuilder method*), 190
visit_message() (*robot.model.visitor.SuiteVisitor method*), 192
visit_message() (*robot.output.console.dotted.StatusReporter method*), 193
visit_message() (*robot.output.xmllogger.XmlLogger method*), 204
visit_message() (*robot.reporting.outputwriter.OutputWriter method*), 223
visit_message() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 226
visit_message() (*robot.result.configurer.SuiteConfigurer method*), 229
visit_message() (*robot.result.keywordremover.AllKeywordsRemover method*), 232
visit_message() (*robot.result.keywordremover.ByNameKeywordRemover method*), 234
visit_message() (*robot.result.keywordremover.ByTagKeywordRemover method*), 235
visit_message() (*robot.result.keywordremover.ForLoopItemsRemover method*), 236
visit_message() (*robot.result.keywordremover.PassedKeywordRemover method*), 233
visit_message() (*robot.result.keywordremover.WaitUntilKeywordSucceedsRemover method*), 237

visit_message() (robot.result.keywordremover.WarningAndErrorFinder method), 238
visit_message() (robot.result.merger.Merger method), 239
visit_message() (robot.result.messagefilter.MessageFilter method), 240
visit_message() (robot.result.resultbuilder.RemoveKeywords method), 249
visit_message() (robot.result.suiteteardownfailed.SuiteTeardownFailed method), 250
visit_message() (robot.result.suiteteardownfailed.SuiteTeardownFailureHandler method), 250
visit_message() (robot.result.visitor.ResultVisitor method), 252
visit_message() (robot.running.randomizer.Randomizer method), 274
visit_message() (robot.running.runner.Runner method), 275
visit_result() (robot.output.xmllogger.XmlLogger method), 204
visit_result() (robot.reporting.outputwriter.OutputWriter method), 223
visit_result() (robot.reporting.xunitwriter.XUnitFileWriter method), 226
visit_result() (robot.result.visitor.ResultVisitor method), 251
visit_stat() (robot.output.xmllogger.XmlLogger method), 203
visit_stat() (robot.reporting.outputwriter.OutputWriter method), 223
visit_stat() (robot.reporting.xunitwriter.XUnitFileWriter method), 226
visit_stat() (robot.result.visitor.ResultVisitor method), 251
visit_statistics() (robot.output.xmllogger.XmlLogger method), 204
visit_statistics() (robot.reporting.outputwriter.OutputWriter method), 223
visit_statistics() (robot.reporting.xunitwriter.XUnitFileWriter method), 225
visit_statistics() (robot.result.visitor.ResultVisitor method), 251
visit_suite() (robot.conf.gatherfailed.GatherFailedSuites method), 17
visit_suite() (robot.conf.gatherfailed.GatherFailedTests method), 16
visit_suite() (robot.model.configurer.SuiteConfigurer method), 170
visit_suite() (robot.model.filter.EmptySuiteRemover method), 172
visit_suite() (robot.model.filter.Filter method), 173
visit_suite() (robot.model.modifier.ModelModifier method), 178
visit_suite() (robot.model.statistics.StatisticsBuilder method), 180
visit_suite() (robot.model.tagsetter.TagSetter method), 185
visit_suite() (robot.model.totalstatistics.TotalStatisticsBuilder method), 190
visit_suite() (robot.model.visitor.SuiteVisitor method), 191
visit_suite() (robot.output.console.dotted.StatusReporter method), 193
visit_suite() (robot.output.xmllogger.XmlLogger method), 204
visit_suite() (robot.reporting.outputwriter.OutputWriter method), 224
visit_suite() (robot.reporting.xunitwriter.XUnitFileWriter method), 226
visit_suite() (robot.result.configurer.SuiteConfigurer method), 228
visit_suite() (robot.result.keywordremover.AllKeywordsRemover method), 233
visit_suite() (robot.result.keywordremover.ByNameKeywordRemover method), 234
visit_suite() (robot.result.keywordremover.ByTagKeywordRemover method), 235
visit_suite() (robot.result.keywordremover.ForLoopItemsRemover method), 236
visit_suite() (robot.result.keywordremover.PassedKeywordRemover method), 234
visit_suite() (robot.result.keywordremover.WaitUntilKeywordSucceed method), 237
visit_suite() (robot.result.keywordremover.WarningAndErrorFinder method), 238
visit_suite() (robot.result.merger.Merger method), 239
visit_suite() (robot.result.messagefilter.MessageFilter method), 240
visit_suite() (robot.result.resultbuilder.RemoveKeywords method), 249
visit_suite() (robot.result.suiteteardownfailed.SuiteTeardownFailed method), 251
visit_suite() (robot.result.suiteteardownfailed.SuiteTeardownFailureHandler method), 250
visit_suite() (robot.result.visitor.ResultVisitor method), 252
visit_suite() (robot.running.randomizer.Randomizer method), 274
visit_suite() (robot.running.runner.Runner method), 275
visit_suite_statistics() (robot.output.xmllogger.XmlLogger method),

204
visit_suite_statistics() (*robot.reporting.outputwriter.OutputWriter method*), 224
visit_suite_statistics() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 226
visit_suite_statistics() (*robot.result.visitor.ResultVisitor method*), 251
visit_tag_statistics() (*robot.output.xmllogger.XmlLogger method*), 204
visit_tag_statistics() (*robot.reporting.outputwriter.OutputWriter method*), 224
visit_tag_statistics() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 226
visit_tag_statistics() (*robot.result.visitor.ResultVisitor method*), 251
visit_test() (*robot.conf.gatherfailed.GatherFailedSuites method*), 16
visit_test() (*robot.conf.gatherfailed.GatherFailedTests method*), 15
visit_test() (*robot.model.configurer.SuiteConfigurer method*), 171
visit_test() (*robot.model.filter.EmptySuiteRemover method*), 171
visit_test() (*robot.model.filter.Filter method*), 173
visit_test() (*robot.model.modifier.ModelModifier method*), 179
visit_test() (*robot.model.statistics.StatisticsBuilder method*), 180
visit_test() (*robot.model.tagsetter.TagSetter method*), 184
visit_test() (*robot.model.totalstatistics.TotalStatisticsBuilder method*), 189
visit_test() (*robot.model.visitor.SuiteVisitor method*), 191
visit_test() (*robot.output.console.dotted.StatusReporter method*), 193
visit_test() (*robot.output.xmllogger.XmlLogger method*), 204
visit_test() (*robot.reporting.outputwriter.OutputWriter method*), 224
visit_test() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 225
visit_test() (*robot.result.configurer.SuiteConfigurer method*), 229
visit_test() (*robot.result.keywordremover.AllKeywordsRemover method*), 233
visit_test() (*robot.result.keywordremover.ByNameKeywordRemover method*), 234
visit_test() (*robot.result.keywordremover.ByTagKeywordRemover method*), 235
visit_test() (*robot.result.keywordremover.ForLoopItemsRemover method*), 236
visit_test() (*robot.result.keywordremover.PassedKeywordRemover method*), 233
visit_test() (*robot.result.keywordremover.WaitUntilKeywordSucceeds method*), 237
visit_test() (*robot.result.keywordremover.WarningAndErrorFinder method*), 238
visit_test() (*robot.result.merger.Merger method*), 239
visit_test() (*robot.result.messagefilter.MessageFilter method*), 240
visit_test() (*robot.result.resultbuilder.RemoveKeywords method*), 248
visit_test() (*robot.result.suiteteardownfailed.SuiteTeardownFailed method*), 250
visit_test() (*robot.result.suiteteardownfailed.SuiteTeardownFailureH method*), 249
visit_test() (*robot.result.visitor.ResultVisitor method*), 252
visit_test() (*robot.running.randomizer.Randomizer method*), 274
visit_test() (*robot.running.runner.Runner method*), 275
visit_total_statistics() (*robot.output.xmllogger.XmlLogger method*), 204
visit_total_statistics() (*robot.reporting.outputwriter.OutputWriter method*), 224
visit_total_statistics() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 227
visit_total_statistics() (*robot.result.visitor.ResultVisitor method*), 251

W

wait_for_process() (*robot.libraries.Process.Process method*), 72
wait_until_created() (*robot.libraries.OperatingSystem.OperatingSystem method*), 63
wait_until_keyword_succeeds() (*robot.libraries.BuiltIn.BuiltIn method*), 46
wait_until_removed() (*robot.libraries.OperatingSystem.OperatingSystem method*), 63

wait_variable() (*robot.libraries.dialogs_py.InputDialog method*), 126
wait_variable() (*robot.libraries.dialogs_py.MessageDialog method*), 113
wait_variable() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 152
wait_variable() (*robot.libraries.dialogs_py.PassFailDialog method*), 165
wait_variable() (*robot.libraries.dialogs_py.SelectionDialog method*), 139
wait_visibility() (*robot.libraries.dialogs_py.InputDialog method*), 126
wait_visibility() (*robot.libraries.dialogs_py.MessageDialog method*), 113
wait_visibility() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 152
wait_visibility() (*robot.libraries.dialogs_py.PassFailDialog method*), 165
wait_visibility() (*robot.libraries.dialogs_py.SelectionDialog method*), 139
wait_window() (*robot.libraries.dialogs_py.InputDialog method*), 126
wait_window() (*robot.libraries.dialogs_py.MessageDialog method*), 113
wait_window() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 152
wait_window() (*robot.libraries.dialogs_py.PassFailDialog method*), 165
wait_window() (*robot.libraries.dialogs_py.SelectionDialog method*), 139
WaitUntilKeywordSucceedsRemover (*class in robot.result.keywordremover*), 236
waitvar() (*robot.libraries.dialogs_py.InputDialog method*), 126
waitvar() (*robot.libraries.dialogs_py.MessageDialog method*), 113
waitvar() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 152
waitvar() (*robot.libraries.dialogs_py.PassFailDialog method*), 165
waitvar() (*robot.libraries.dialogs_py.SelectionDialog method*), 139
warn() (*in module robot.api.logger*), 14
warn() (*in module robot.output.librarylogger*), 196
warn() (*robot.output.filelogger.FileLogger method*), 196
warn() (*robot.output.logger.Logger method*), 199
warn() (*robot.output.loggerhelper.AbstractLogger method*), 200
warn() (*robot.output.output.Output method*), 201
warning() (*robot.parsing.restsupport.CaptureRobotData method*), 214
WarningAndErrorFinder (*class in robot.result.keywordremover*), 237
winfo_atom() (*robot.libraries.dialogs_py.InputDialog method*), 126
winfo_atom() (*robot.libraries.dialogs_py.MessageDialog method*), 113
winfo_atom() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 152
winfo_atom() (*robot.libraries.dialogs_py.PassFailDialog method*), 165
winfo_atom() (*robot.libraries.dialogs_py.SelectionDialog method*), 139
winfo_atomname() (*robot.libraries.dialogs_py.InputDialog method*), 126
winfo_atomname() (*robot.libraries.dialogs_py.MessageDialog method*), 113
winfo_atomname() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 152
winfo_atomname() (*robot.libraries.dialogs_py.PassFailDialog method*), 165
winfo_atomname() (*robot.libraries.dialogs_py.SelectionDialog method*), 139
winfo_cells() (*robot.libraries.dialogs_py.InputDialog method*), 126
winfo_cells() (*robot.libraries.dialogs_py.MessageDialog method*), 113
winfo_cells() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 152
winfo_cells() (*robot.libraries.dialogs_py.PassFailDialog method*), 165
winfo_cells() (*robot.libraries.dialogs_py.SelectionDialog method*), 139
winfo_children() (*robot.libraries.dialogs_py.InputDialog method*), 126
winfo_children() (*robot.libraries.dialogs_py.MessageDialog method*), 113
winfo_children() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 152
winfo_children() (*robot.libraries.dialogs_py.PassFailDialog method*), 165
winfo_children() (*robot.libraries.dialogs_py.SelectionDialog method*), 139
winfo_class() (*robot.libraries.dialogs_py.InputDialog method*), 126
winfo_class() (*robot.libraries.dialogs_py.MessageDialog method*), 113
winfo_class() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 152
winfo_class() (*robot.libraries.dialogs_py.PassFailDialog method*), 165
winfo_class() (*robot.libraries.dialogs_py.SelectionDialog*

method), 139
winfo_colormapfull() (*robot.libraries.dialogs_py.InputDialog method*), 126
winfo_colormapfull() (*robot.libraries.dialogs_py.MessageDialog method*), 113
winfo_colormapfull() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 152
winfo_colormapfull() (*robot.libraries.dialogs_py.PassFailDialog method*), 165
winfo_colormapfull() (*robot.libraries.dialogs_py.SelectionDialog method*), 139
winfo_containing() (*robot.libraries.dialogs_py.InputDialog method*), 126
winfo_containing() (*robot.libraries.dialogs_py.MessageDialog method*), 113
winfo_containing() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 152
winfo_containing() (*robot.libraries.dialogs_py.PassFailDialog method*), 165
winfo_containing() (*robot.libraries.dialogs_py.SelectionDialog method*), 139
winfo_depth() (*robot.libraries.dialogs_py.InputDialog method*), 126
winfo_depth() (*robot.libraries.dialogs_py.MessageDialog method*), 113
winfo_depth() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 152
winfo_depth() (*robot.libraries.dialogs_py.PassFailDialog method*), 165
winfo_depth() (*robot.libraries.dialogs_py.SelectionDialog method*), 139
winfo_exists() (*robot.libraries.dialogs_py.InputDialog method*), 126
winfo_exists() (*robot.libraries.dialogs_py.MessageDialog method*), 113
winfo_exists() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 152
winfo_exists() (*robot.libraries.dialogs_py.PassFailDialog method*), 165
winfo_exists() (*robot.libraries.dialogs_py.SelectionDialog method*), 139
winfo_fpixels() (*robot.libraries.dialogs_py.InputDialog method*), 126
winfo_fpixels() (*robot.libraries.dialogs_py.MessageDialog method*), 113
winfo_fpixels() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 152
winfo_fpixels() (*robot.libraries.dialogs_py.PassFailDialog method*), 165
winfo_fpixels() (*robot.libraries.dialogs_py.SelectionDialog method*), 139
winfo_geometry() (*robot.libraries.dialogs_py.InputDialog method*), 126
winfo_geometry() (*robot.libraries.dialogs_py.MessageDialog method*), 113
winfo_geometry() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 152
winfo_geometry() (*robot.libraries.dialogs_py.PassFailDialog method*), 165
winfo_geometry() (*robot.libraries.dialogs_py.SelectionDialog method*), 139
winfo_height() (*robot.libraries.dialogs_py.InputDialog method*), 126
winfo_height() (*robot.libraries.dialogs_py.MessageDialog method*), 113
winfo_height() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 152
winfo_height() (*robot.libraries.dialogs_py.PassFailDialog method*), 165
winfo_height() (*robot.libraries.dialogs_py.SelectionDialog method*), 139
winfo_id() (*robot.libraries.dialogs_py.InputDialog method*), 126
winfo_id() (*robot.libraries.dialogs_py.MessageDialog method*), 113
winfo_id() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 152
winfo_id() (*robot.libraries.dialogs_py.PassFailDialog method*), 165
winfo_id() (*robot.libraries.dialogs_py.SelectionDialog method*), 139
winfo_interps() (*robot.libraries.dialogs_py.InputDialog method*), 127
winfo_interps() (*robot.libraries.dialogs_py.MessageDialog method*), 114
winfo_interps() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 153
winfo_interps() (*robot.libraries.dialogs_py.PassFailDialog method*), 166
winfo_interps() (*robot.libraries.dialogs_py.SelectionDialog method*), 140
winfo_ismapped() (*robot.libraries.dialogs_py.InputDialog method*), 127
winfo_ismapped() (*robot.libraries.dialogs_py.MessageDialog method*), 114
winfo_ismapped() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 153
winfo_ismapped() (*robot.libraries.dialogs_py.PassFailDialog*

method), 166
winfo_ismapped() (*robot.libraries.dialogs_py.SelectionDialog method*), 140
winfo_manager() (*robot.libraries.dialogs_py.InputDialog method*), 127
winfo_manager() (*robot.libraries.dialogs_py.MessageDialog method*), 114
winfo_manager() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 153
winfo_manager() (*robot.libraries.dialogs_py.PassFailDialog method*), 166
winfo_manager() (*robot.libraries.dialogs_py.SelectionDialog method*), 140
winfo_name() (*robot.libraries.dialogs_py.InputDialog method*), 127
winfo_name() (*robot.libraries.dialogs_py.MessageDialog method*), 114
winfo_name() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 153
winfo_name() (*robot.libraries.dialogs_py.PassFailDialog method*), 166
winfo_name() (*robot.libraries.dialogs_py.SelectionDialog method*), 140
winfo_parent() (*robot.libraries.dialogs_py.InputDialog method*), 127
winfo_parent() (*robot.libraries.dialogs_py.MessageDialog method*), 114
winfo_parent() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 153
winfo_parent() (*robot.libraries.dialogs_py.PassFailDialog method*), 166
winfo_parent() (*robot.libraries.dialogs_py.SelectionDialog method*), 140
winfo_pathname() (*robot.libraries.dialogs_py.InputDialog method*), 127
winfo_pathname() (*robot.libraries.dialogs_py.MessageDialog method*), 114
winfo_pathname() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 153
winfo_pathname() (*robot.libraries.dialogs_py.PassFailDialog method*), 166
winfo_pathname() (*robot.libraries.dialogs_py.SelectionDialog method*), 140
winfo_pixels() (*robot.libraries.dialogs_py.InputDialog method*), 127
winfo_pixels() (*robot.libraries.dialogs_py.MessageDialog method*), 114
winfo_pixels() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 153
winfo_pixels() (*robot.libraries.dialogs_py.PassFailDialog method*), 166
winfo_pixels() (*robot.libraries.dialogs_py.SelectionDialog method*), 140
winfo_pointerx() (*robot.libraries.dialogs_py.InputDialog method*), 127
winfo_pointerx() (*robot.libraries.dialogs_py.MessageDialog method*), 114
winfo_pointerx() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 153
winfo_pointerx() (*robot.libraries.dialogs_py.PassFailDialog method*), 166
winfo_pointerx() (*robot.libraries.dialogs_py.SelectionDialog method*), 140
winfo_pointerxy() (*robot.libraries.dialogs_py.InputDialog method*), 127
winfo_pointerxy() (*robot.libraries.dialogs_py.MessageDialog method*), 114
winfo_pointerxy() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 153
winfo_pointerxy() (*robot.libraries.dialogs_py.PassFailDialog method*), 166
winfo_pointerxy() (*robot.libraries.dialogs_py.SelectionDialog method*), 140
winfo_pointery() (*robot.libraries.dialogs_py.InputDialog method*), 127
winfo_pointery() (*robot.libraries.dialogs_py.MessageDialog method*), 114
winfo_pointery() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 153
winfo_pointery() (*robot.libraries.dialogs_py.PassFailDialog method*), 166
winfo_pointery() (*robot.libraries.dialogs_py.SelectionDialog method*), 140
winfo_reqheight() (*robot.libraries.dialogs_py.InputDialog method*), 127
winfo_reqheight() (*robot.libraries.dialogs_py.MessageDialog method*), 114
winfo_reqheight() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 153
winfo_reqheight() (*robot.libraries.dialogs_py.PassFailDialog method*), 166
winfo_reqheight() (*robot.libraries.dialogs_py.SelectionDialog method*), 140
winfo_reqwidth() (*robot.libraries.dialogs_py.InputDialog method*), 127
winfo_reqwidth() (*robot.libraries.dialogs_py.MessageDialog method*), 114
winfo_reqwidth() (*robot.libraries.dialogs_py.MultipleSelectionDialog*

method), 153
winfo_reqwidth() (*robot.libraries.dialogs_py.PassFailDialog method*), 166
winfo_reqwidth() (*robot.libraries.dialogs_py.SelectionDialog method*), 140
winfo_rgb() (*robot.libraries.dialogs_py.InputDialog method*), 127
winfo_rgb() (*robot.libraries.dialogs_py.MessageDialog method*), 114
winfo_rgb() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 153
winfo_rgb() (*robot.libraries.dialogs_py.PassFailDialog method*), 166
winfo_rgb() (*robot.libraries.dialogs_py.SelectionDialog method*), 140
winfo_rootx() (*robot.libraries.dialogs_py.InputDialog method*), 127
winfo_rootx() (*robot.libraries.dialogs_py.MessageDialog method*), 114
winfo_rootx() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 153
winfo_rootx() (*robot.libraries.dialogs_py.PassFailDialog method*), 166
winfo_rootx() (*robot.libraries.dialogs_py.SelectionDialog method*), 140
winfo_rooty() (*robot.libraries.dialogs_py.InputDialog method*), 127
winfo_rooty() (*robot.libraries.dialogs_py.MessageDialog method*), 114
winfo_rooty() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 153
winfo_rooty() (*robot.libraries.dialogs_py.PassFailDialog method*), 166
winfo_rooty() (*robot.libraries.dialogs_py.SelectionDialog method*), 140
winfo_screen() (*robot.libraries.dialogs_py.InputDialog method*), 127
winfo_screen() (*robot.libraries.dialogs_py.MessageDialog method*), 114
winfo_screen() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 153
winfo_screen() (*robot.libraries.dialogs_py.PassFailDialog method*), 166
winfo_screen() (*robot.libraries.dialogs_py.SelectionDialog method*), 140
winfo_screencells() (*robot.libraries.dialogs_py.InputDialog method*), 127
winfo_screencells() (*robot.libraries.dialogs_py.MessageDialog method*), 114
winfo_screencells() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 153
winfo_screencells() (*robot.libraries.dialogs_py.PassFailDialog method*), 166
winfo_screencells() (*robot.libraries.dialogs_py.SelectionDialog method*), 140
winfo_screendepth() (*robot.libraries.dialogs_py.InputDialog method*), 127
winfo_screendepth() (*robot.libraries.dialogs_py.MessageDialog method*), 114
winfo_screendepth() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 153
winfo_screendepth() (*robot.libraries.dialogs_py.PassFailDialog method*), 166
winfo_screendepth() (*robot.libraries.dialogs_py.SelectionDialog method*), 140
winfo_screenheight() (*robot.libraries.dialogs_py.InputDialog method*), 127
winfo_screenheight() (*robot.libraries.dialogs_py.MessageDialog method*), 114
winfo_screenheight() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 153
winfo_screenheight() (*robot.libraries.dialogs_py.PassFailDialog method*), 166
winfo_screenheight() (*robot.libraries.dialogs_py.SelectionDialog method*), 140
winfo_screenmmheight() (*robot.libraries.dialogs_py.InputDialog method*), 127
winfo_screenmmheight() (*robot.libraries.dialogs_py.MessageDialog method*), 114
winfo_screenmmheight() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 153
winfo_screenmmheight() (*robot.libraries.dialogs_py.PassFailDialog method*), 166
winfo_screenmmheight() (*robot.libraries.dialogs_py.SelectionDialog method*), 140
winfo_screenmmwidth() (*robot.libraries.dialogs_py.InputDialog method*), 127

winfo_screenmmwidth() (*robot.libraries.dialogs_py.MessageDialog method*), 114
winfo_screenmmwidth() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 153
winfo_screenmmwidth() (*robot.libraries.dialogs_py.PassFailDialog method*), 166
winfo_screenmmwidth() (*robot.libraries.dialogs_py.SelectionDialog method*), 140
winfo_screenvisual() (*robot.libraries.dialogs_py.InputDialog method*), 128
winfo_screenvisual() (*robot.libraries.dialogs_py.MessageDialog method*), 115
winfo_screenvisual() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 154
winfo_screenvisual() (*robot.libraries.dialogs_py.PassFailDialog method*), 167
winfo_screenvisual() (*robot.libraries.dialogs_py.SelectionDialog method*), 141
winfo_screenwidth() (*robot.libraries.dialogs_py.InputDialog method*), 128
winfo_screenwidth() (*robot.libraries.dialogs_py.MessageDialog method*), 115
winfo_screenwidth() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 154
winfo_screenwidth() (*robot.libraries.dialogs_py.PassFailDialog method*), 167
winfo_screenwidth() (*robot.libraries.dialogs_py.SelectionDialog method*), 141
winfo_server() (*robot.libraries.dialogs_py.InputDialog method*), 128
winfo_server() (*robot.libraries.dialogs_py.MessageDialog method*), 115
winfo_server() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 154
winfo_server() (*robot.libraries.dialogs_py.PassFailDialog method*), 167
winfo_server() (*robot.libraries.dialogs_py.SelectionDialog method*), 141
winfo_toplevel() (*robot.libraries.dialogs_py.InputDialog method*), 128
winfo_toplevel() (*robot.libraries.dialogs_py.MessageDialog method*), 115
winfo_toplevel() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 154
winfo_toplevel() (*robot.libraries.dialogs_py.PassFailDialog method*), 167
winfo_toplevel() (*robot.libraries.dialogs_py.SelectionDialog method*), 141
winfo_viewable() (*robot.libraries.dialogs_py.InputDialog method*), 128
winfo_viewable() (*robot.libraries.dialogs_py.MessageDialog method*), 115
winfo_viewable() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 154
winfo_viewable() (*robot.libraries.dialogs_py.PassFailDialog method*), 167
winfo_viewable() (*robot.libraries.dialogs_py.SelectionDialog method*), 141
winfo_visual() (*robot.libraries.dialogs_py.InputDialog method*), 128
winfo_visual() (*robot.libraries.dialogs_py.MessageDialog method*), 115
winfo_visual() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 154
winfo_visual() (*robot.libraries.dialogs_py.PassFailDialog method*), 167
winfo_visual() (*robot.libraries.dialogs_py.SelectionDialog method*), 141
winfo_visualid() (*robot.libraries.dialogs_py.InputDialog method*), 128
winfo_visualid() (*robot.libraries.dialogs_py.MessageDialog method*), 115
winfo_visualid() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 154
winfo_visualid() (*robot.libraries.dialogs_py.PassFailDialog method*), 167
winfo_visualid() (*robot.libraries.dialogs_py.SelectionDialog method*), 141
winfo_visualsavailable() (*robot.libraries.dialogs_py.InputDialog method*), 128
winfo_visualsavailable() (*robot.libraries.dialogs_py.MessageDialog method*), 115
winfo_visualsavailable() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 154
winfo_visualsavailable() (*robot.libraries.dialogs_py.PassFailDialog method*), 167
winfo_visualsavailable() (*robot.libraries.dialogs_py.SelectionDialog method*), 141
winfo_vrootheight()

(*robot.libraries.dialogs_py.InputDialog method*), 128
winfo_vrootheight() (*robot.libraries.dialogs_py.MessageDialog method*), 115
winfo_vrootheight() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 154
winfo_vrootheight() (*robot.libraries.dialogs_py.PassFailDialog method*), 167
winfo_vrootheight() (*robot.libraries.dialogs_py.SelectionDialog method*), 141
winfo_vrootwidth() (*robot.libraries.dialogs_py.InputDialog method*), 128
winfo_vrootwidth() (*robot.libraries.dialogs_py.MessageDialog method*), 115
winfo_vrootwidth() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 154
winfo_vrootwidth() (*robot.libraries.dialogs_py.PassFailDialog method*), 167
winfo_vrootwidth() (*robot.libraries.dialogs_py.SelectionDialog method*), 141
winfo_vrootx() (*robot.libraries.dialogs_py.InputDialog method*), 128
winfo_vrootx() (*robot.libraries.dialogs_py.MessageDialog method*), 115
winfo_vrootx() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 154
winfo_vrootx() (*robot.libraries.dialogs_py.PassFailDialog method*), 167
winfo_vrootx() (*robot.libraries.dialogs_py.SelectionDialog method*), 141
winfo_vrooty() (*robot.libraries.dialogs_py.InputDialog method*), 128
winfo_vrooty() (*robot.libraries.dialogs_py.MessageDialog method*), 115
winfo_vrooty() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 154
winfo_vrooty() (*robot.libraries.dialogs_py.PassFailDialog method*), 167
winfo_vrooty() (*robot.libraries.dialogs_py.SelectionDialog method*), 141
winfo_width() (*robot.libraries.dialogs_py.InputDialog method*), 128
winfo_width() (*robot.libraries.dialogs_py.MessageDialog method*), 115
winfo_width() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 154
winfo_width() (*robot.libraries.dialogs_py.PassFailDialog method*), 167
winfo_width() (*robot.libraries.dialogs_py.SelectionDialog method*), 141
winfo_x() (*robot.libraries.dialogs_py.InputDialog method*), 128
winfo_x() (*robot.libraries.dialogs_py.MessageDialog method*), 115
winfo_x() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 154
winfo_x() (*robot.libraries.dialogs_py.PassFailDialog method*), 167
winfo_x() (*robot.libraries.dialogs_py.SelectionDialog method*), 141
winfo_y() (*robot.libraries.dialogs_py.InputDialog method*), 128
winfo_y() (*robot.libraries.dialogs_py.MessageDialog method*), 115
winfo_y() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 154
winfo_y() (*robot.libraries.dialogs_py.PassFailDialog method*), 167
winfo_y() (*robot.libraries.dialogs_py.SelectionDialog method*), 141
with_metaclass() (*in module robot.utils.compat*), 284
withdraw() (*robot.libraries.dialogs_py.InputDialog method*), 128
withdraw() (*robot.libraries.dialogs_py.MessageDialog method*), 115
withdraw() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 154
withdraw() (*robot.libraries.dialogs_py.PassFailDialog method*), 167
withdraw() (*robot.libraries.dialogs_py.SelectionDialog method*), 141
wm_aspect() (*robot.libraries.dialogs_py.InputDialog method*), 128
wm_aspect() (*robot.libraries.dialogs_py.MessageDialog method*), 115
wm_aspect() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 154
wm_aspect() (*robot.libraries.dialogs_py.PassFailDialog method*), 167
wm_aspect() (*robot.libraries.dialogs_py.SelectionDialog method*), 141
wm_attributes() (*robot.libraries.dialogs_py.InputDialog method*), 129
wm_attributes() (*robot.libraries.dialogs_py.MessageDialog method*), 116
wm_attributes() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 155
wm_attributes() (*robot.libraries.dialogs_py.PassFailDialog*

method), 168
wm_attributes() (*robot.libraries.dialogs_py.SelectionDialog method*), 142
wm_client() (*robot.libraries.dialogs_py.InputDialog method*), 129
wm_client() (*robot.libraries.dialogs_py.MessageDialog method*), 116
wm_client() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 155
wm_client() (*robot.libraries.dialogs_py.PassFailDialog method*), 168
wm_client() (*robot.libraries.dialogs_py.SelectionDialog method*), 142
wm_colormapwindows() (*robot.libraries.dialogs_py.InputDialog method*), 129
wm_colormapwindows() (*robot.libraries.dialogs_py.MessageDialog method*), 116
wm_colormapwindows() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 155
wm_colormapwindows() (*robot.libraries.dialogs_py.PassFailDialog method*), 168
wm_colormapwindows() (*robot.libraries.dialogs_py.SelectionDialog method*), 142
wm_command() (*robot.libraries.dialogs_py.InputDialog method*), 129
wm_command() (*robot.libraries.dialogs_py.MessageDialog method*), 116
wm_command() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 155
wm_command() (*robot.libraries.dialogs_py.PassFailDialog method*), 168
wm_command() (*robot.libraries.dialogs_py.SelectionDialog method*), 142
wm_deiconify() (*robot.libraries.dialogs_py.InputDialog method*), 129
wm_deiconify() (*robot.libraries.dialogs_py.MessageDialog method*), 116
wm_deiconify() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 155
wm_deiconify() (*robot.libraries.dialogs_py.PassFailDialog method*), 168
wm_deiconify() (*robot.libraries.dialogs_py.SelectionDialog method*), 142
wm_focusmodel() (*robot.libraries.dialogs_py.InputDialog method*), 129
wm_focusmodel() (*robot.libraries.dialogs_py.MessageDialog method*), 116
wm_focusmodel() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 155
wm_focusmodel() (*robot.libraries.dialogs_py.PassFailDialog method*), 168
wm_focusmodel() (*robot.libraries.dialogs_py.SelectionDialog method*), 142
wm_frame() (*robot.libraries.dialogs_py.InputDialog method*), 129
wm_frame() (*robot.libraries.dialogs_py.MessageDialog method*), 116
wm_frame() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 155
wm_frame() (*robot.libraries.dialogs_py.PassFailDialog method*), 168
wm_frame() (*robot.libraries.dialogs_py.SelectionDialog method*), 142
wm_geometry() (*robot.libraries.dialogs_py.InputDialog method*), 129
wm_geometry() (*robot.libraries.dialogs_py.MessageDialog method*), 116
wm_geometry() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 155
wm_geometry() (*robot.libraries.dialogs_py.PassFailDialog method*), 168
wm_geometry() (*robot.libraries.dialogs_py.SelectionDialog method*), 142
wm_grid() (*robot.libraries.dialogs_py.InputDialog method*), 129
wm_grid() (*robot.libraries.dialogs_py.MessageDialog method*), 116
wm_grid() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 155
wm_grid() (*robot.libraries.dialogs_py.PassFailDialog method*), 168
wm_grid() (*robot.libraries.dialogs_py.SelectionDialog method*), 142
wm_group() (*robot.libraries.dialogs_py.InputDialog method*), 129
wm_group() (*robot.libraries.dialogs_py.MessageDialog method*), 116
wm_group() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 155
wm_group() (*robot.libraries.dialogs_py.PassFailDialog method*), 168
wm_group() (*robot.libraries.dialogs_py.SelectionDialog method*), 142
wm_iconbitmap() (*robot.libraries.dialogs_py.InputDialog method*), 129
wm_iconbitmap() (*robot.libraries.dialogs_py.MessageDialog method*), 116
wm_iconbitmap() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 155
wm_iconbitmap() (*robot.libraries.dialogs_py.PassFailDialog method*), 168
wm_iconbitmap() (*robot.libraries.dialogs_py.SelectionDialog method*), 142

wm_iconify() (*robot.libraries.dialogs_py.InputDialog method*), 129
wm_iconify() (*robot.libraries.dialogs_py.MessageDialog method*), 116
wm_iconify() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 155
wm_iconify() (*robot.libraries.dialogs_py.PassFailDialog method*), 168
wm_iconify() (*robot.libraries.dialogs_py.SelectionDialog method*), 142
wm_iconmask() (*robot.libraries.dialogs_py.InputDialog method*), 129
wm_iconmask() (*robot.libraries.dialogs_py.MessageDialog method*), 116
wm_iconmask() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 155
wm_iconmask() (*robot.libraries.dialogs_py.PassFailDialog method*), 168
wm_iconmask() (*robot.libraries.dialogs_py.SelectionDialog method*), 142
wm_iconname() (*robot.libraries.dialogs_py.InputDialog method*), 130
wm_iconname() (*robot.libraries.dialogs_py.MessageDialog method*), 117
wm_iconname() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 156
wm_iconname() (*robot.libraries.dialogs_py.PassFailDialog method*), 169
wm_iconname() (*robot.libraries.dialogs_py.SelectionDialog method*), 143
wm_iconposition() (*robot.libraries.dialogs_py.InputDialog method*), 130
wm_iconposition() (*robot.libraries.dialogs_py.MessageDialog method*), 117
wm_iconposition() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 156
wm_iconposition() (*robot.libraries.dialogs_py.PassFailDialog method*), 169
wm_iconposition() (*robot.libraries.dialogs_py.SelectionDialog method*), 143
wm_iconwindow() (*robot.libraries.dialogs_py.InputDialog method*), 130
wm_iconwindow() (*robot.libraries.dialogs_py.MessageDialog method*), 117
wm_iconwindow() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 156
wm_iconwindow() (*robot.libraries.dialogs_py.PassFailDialog method*), 169
wm_iconwindow() (*robot.libraries.dialogs_py.SelectionDialog method*), 143
wm_maxsize() (*robot.libraries.dialogs_py.InputDialog method*), 130
wm_maxsize() (*robot.libraries.dialogs_py.MessageDialog method*), 117
wm_maxsize() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 156
wm_maxsize() (*robot.libraries.dialogs_py.PassFailDialog method*), 169
wm_maxsize() (*robot.libraries.dialogs_py.SelectionDialog method*), 143
wm_minsize() (*robot.libraries.dialogs_py.InputDialog method*), 130
wm_minsize() (*robot.libraries.dialogs_py.MessageDialog method*), 117
wm_minsize() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 156
wm_minsize() (*robot.libraries.dialogs_py.PassFailDialog method*), 169
wm_minsize() (*robot.libraries.dialogs_py.SelectionDialog method*), 143
wm_overrideredirect() (*robot.libraries.dialogs_py.InputDialog method*), 130
wm_overrideredirect() (*robot.libraries.dialogs_py.MessageDialog method*), 117
wm_overrideredirect() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 156
wm_overrideredirect() (*robot.libraries.dialogs_py.PassFailDialog method*), 169
wm_overrideredirect() (*robot.libraries.dialogs_py.SelectionDialog method*), 143
wm_positionfrom() (*robot.libraries.dialogs_py.InputDialog method*), 130
wm_positionfrom() (*robot.libraries.dialogs_py.MessageDialog method*), 117
wm_positionfrom() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 156
wm_positionfrom() (*robot.libraries.dialogs_py.PassFailDialog method*), 169
wm_positionfrom() (*robot.libraries.dialogs_py.SelectionDialog method*), 143
wm_protocol() (*robot.libraries.dialogs_py.InputDialog method*), 130
wm_protocol() (*robot.libraries.dialogs_py.MessageDialog*

method), 117
wm_protocol() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 156
wm_protocol() (*robot.libraries.dialogs_py.PassFailDialog method*), 169
wm_protocol() (*robot.libraries.dialogs_py.SelectionDialog method*), 143
wm_resizable() (*robot.libraries.dialogs_py.InputDialog method*), 130
wm_resizable() (*robot.libraries.dialogs_py.MessageDialog method*), 117
wm_resizable() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 156
wm_resizable() (*robot.libraries.dialogs_py.PassFailDialog method*), 169
wm_resizable() (*robot.libraries.dialogs_py.SelectionDialog method*), 143
wm_sizefrom() (*robot.libraries.dialogs_py.InputDialog method*), 130
wm_sizefrom() (*robot.libraries.dialogs_py.MessageDialog method*), 117
wm_sizefrom() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 156
wm_sizefrom() (*robot.libraries.dialogs_py.PassFailDialog method*), 169
wm_sizefrom() (*robot.libraries.dialogs_py.SelectionDialog method*), 143
wm_state() (*robot.libraries.dialogs_py.InputDialog method*), 130
wm_state() (*robot.libraries.dialogs_py.MessageDialog method*), 117
wm_state() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 156
wm_state() (*robot.libraries.dialogs_py.PassFailDialog method*), 169
wm_state() (*robot.libraries.dialogs_py.SelectionDialog method*), 143
wm_title() (*robot.libraries.dialogs_py.InputDialog method*), 130
wm_title() (*robot.libraries.dialogs_py.MessageDialog method*), 117
wm_title() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 156
wm_title() (*robot.libraries.dialogs_py.PassFailDialog method*), 169
wm_title() (*robot.libraries.dialogs_py.SelectionDialog method*), 143
wm_transient() (*robot.libraries.dialogs_py.InputDialog method*), 130
wm_transient() (*robot.libraries.dialogs_py.MessageDialog method*), 117
wm_transient() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 156
wm_transient() (*robot.libraries.dialogs_py.PassFailDialog method*), 169
wm_transient() (*robot.libraries.dialogs_py.SelectionDialog method*), 143
wm_withdraw() (*robot.libraries.dialogs_py.InputDialog method*), 130
wm_withdraw() (*robot.libraries.dialogs_py.MessageDialog method*), 117
wm_withdraw() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 156
wm_withdraw() (*robot.libraries.dialogs_py.PassFailDialog method*), 169
wm_withdraw() (*robot.libraries.dialogs_py.SelectionDialog method*), 143
write() (*in module robot.api.logger*), 14
write() (*in module robot.output.librarylogger*), 196
write() (*robot.htmldata.htmlfilewriter.CssFileWriter method*), 19
write() (*robot.htmldata.htmlfilewriter.GeneratorWriter method*), 19
write() (*robot.htmldata.htmlfilewriter.HtmlFileWriter method*), 19
write() (*robot.htmldata.htmlfilewriter.JsFileWriter method*), 19
write() (*robot.htmldata.htmlfilewriter.LineWriter method*), 19
write() (*robot.htmldata.htmlfilewriter.ModelWriter method*), 19
write() (*robot.htmldata.jsonwriter.JsonDumper method*), 20
write() (*robot.htmldata.jsonwriter.JsonWriter method*), 20
write() (*robot.libdocpkg.htmlwriter.LibdocHtmlWriter method*), 21
write() (*robot.libdocpkg.htmlwriter.LibdocModelWriter method*), 21
write() (*robot.libdocpkg.xmlwriter.LibdocXmlWriter method*), 23
write() (*robot.libraries.Telnet.TelnetConnection method*), 89
write() (*robot.output.console.highlighting.HighlightingStream method*), 194
write() (*robot.output.filelogger.FileLogger method*), 196
write() (*robot.output.logger.Logger method*), 199
write() (*robot.output.loggerhelper.AbstractLogger method*), 200
write() (*robot.output.output.Output method*), 201
write() (*robot.reporting.jswriter.JsResultWriter method*), 222
write() (*robot.reporting.jswriter.SplitLogWriter method*), 222
write() (*robot.reporting.jswriter.SuiteWriter method*), 222
write() (*robot.reporting.logreportwriters.LogWriter*

method), 222
write() (*robot.reporting.logreportwriters.ReportWriter method*), 222
write() (*robot.reporting.logreportwriters.RobotModelWriter method*), 222
write() (*robot.reporting.xunitwriter.XUnitWriter method*), 225
write() (*robot.testdoc.TestdocModelWriter method*), 316
write() (*robot.writer.datafilewriter.DataFileWriter method*), 304
write() (*robot.writer.filewriters.HtmlFileWriter method*), 305
write() (*robot.writer.filewriters.PipeSeparatedTxtWriter method*), 305
write() (*robot.writer.filewriters.SpaceSeparatedTxtWriter method*), 305
write() (*robot.writer.filewriters.TsvFileWriter method*), 305
write_bare() (*robot.libraries.Telnet.TelnetConnection method*), 90
write_control_character() (*robot.libraries.Telnet.TelnetConnection method*), 90
write_data() (*robot.libdocpkg.htmlwriter.LibdocModelWriter method*), 21
write_data() (*robot.testdoc.TestdocModelWriter method*), 316
write_json() (*robot.htmldata.jsonwriter.JsonWriter method*), 20
write_results() (*robot.reporting.resultwriter.ResultWriter method*), 224
write_until_expected_output() (*robot.libraries.Telnet.TelnetConnection method*), 90
WritingContext (*class in robot.writer.datafilewriter*), 304

X

XML (*class in robot.libraries.XML*), 93
xml_escape() (*in module robot.utils.markuputils*), 290
XmlElementHandler (*class in robot.result.xmlelementhandlers*), 253
XmlLogger (*class in robot.output.xmllogger*), 203
XmlRpcRemoteClient (*class in robot.libraries.Remote*), 75
XmlWriter (*class in robot.utils.markupwriters*), 290
xunit (*robot.conf.settings.RebotSettings attribute*), 19
xunit (*robot.conf.settings.RobotSettings attribute*), 18
xunit_skip_noncritical (*robot.conf.settings.RebotSettings attribute*), 19
xunit_skip_noncritical (*robot.conf.settings.RobotSettings attribute*), 18
XUnitFileWriter (*class in robot.reporting.xunitwriter*), 225
XUnitWriter (*class in robot.reporting.xunitwriter*), 225

Y

YamlImporter (*class in robot.variables.filesetter*), 298
yellow() (*robot.output.console.highlighting.AnsiHighlighter method*), 194
yellow() (*robot.output.console.highlighting.DosHighlighter method*), 194
yellow() (*robot.output.console.highlighting.NoHighlighting method*), 194