## 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 ....................................................... 303  
Python Module Index .......................................... 305
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.
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.
Java entry points

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

```python
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.
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 internal usage and may change without prior notice.

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

```python
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`, `--escape`, `--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:

```python
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, exit=True)
```

Command line execution entry point for running tests.

**Parameters**

• **arguments** – Command line options and arguments as a list of strings.

• **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:

```python
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, exit=True)

Command line execution entry point for post-processing outputs.

Parameters

- **arguments** – Command line options and arguments as a list of strings.

- **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:

```python
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:

```python
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=())*

Decorator to set custom keyword names and tags to functions and methods.

This decorator creates the **robot_name** and **robot_tags** attributes on the decorated keyword method or function. Robot Framework checks them to determine the keyword’s name and tags, respectively.

**library.py:**

```python
@keyword(name='Login Via User Panel')
def login(username, password):
    # ...
@keyword(name='Logout Via User Panel', tags=['example', 'tags'])
```
def logout():
    # ...

tests.robot:

Login Via User Panel myusername mypassword

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. This feature was added in RF 2.8.7.

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

```python
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'. This is a new feature in RF 2.8.2.

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)
    visit_keyword (kw)

    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)
    visit_test (test)
    visit_keyword (kw)
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
context_type
context_width
context_markers
pre_run_modifiers
run_empty_suite
variables
variable_files
context_colors
critical_tags
flatten_keywords
log
log_level
non_critical_tags
output
output_directory
pre_rebot_modifiers
remove_keywords
report
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_output_config
context_colors
critical_tags
flatten_keywords
log
log_level
non_critical_tags
output
output_directory
pre_rebot_modifiers
process_empty_suite
remove_keywords
report
split_log
statistics_config
status_rc
xunit
xunit_skip_noncritical

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

class robot.htmldata.jartemplate.HtmlTemplate(filename)
    Bases: object

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

4.1. robot package 21
save (output=None, format='HTML')

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

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

```python
robot.libraries.BuiltIn.run_keyword_variant(resolve)
class robot.libraries.BuiltIn.BuiltIn

An always available standard library with often needed keywords.

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

== Table of contents ==
- HTML error messages
- Evaluating expressions
- Boolean arguments
- Multiline string comparisons
- Shortcuts
- Keywords

= HTML error messages =

Many of the keywords accept an optional error message to use if the keyword fails. Starting from Robot Framework 2.8, 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 [https://docs.python.org/2/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 [https://docs.python.org/2/library/os.html|os] and [https://docs.python.org/2/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 either an empty string or case-insensitively equal to false, none or no. 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/2/library/stdtypes.html#truth-value-testing rules as in Python].

True examples:

False examples:

Prior to Robot Framework 2.9 some keywords considered all non-empty strings, including false and no, to be true. Considering none false is new in Robot Framework 3.0.3.

= Multiline string comparisons =

Should Be Equal and Should Be Equal As Strings report the failures using [https://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:

ROBOT_LIBRARY_SCOPE = 'GLOBAL'

ROBOT_LIBRARY_VERSION = '3.1.0.dev1'

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.
New in Robot Framework 2.8.

**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.
New in Robot Framework 2.8.

**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/2/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.
New in Robot Framework 2.8.2.
**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 Hex`.

**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:

- 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/2/tutorial/floatingpoint.html](http://docs.python.org/2/tutorial/floatingpoint.html)

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

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

Converts the given item to an octal string.

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

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

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

**convert_to_string** *(item)*

Converts the given item to a Unicode string.

Uses `__unicode__` or `__str__` method with Python objects and `toString` with Java objects.
Use `Encode String To Bytes` and `Decode Bytes To String` keywords in `String` library if you need to convert between Unicode and byte strings 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`.

This keyword was changed in Robot Framework 2.9 in many ways: - Moved from `Collections` library to `BuiltIn`. - Support also non-string keys in `key=value` syntax. - Returned dictionary is ordered and dot-accessible. - 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. This is a new feature in Robot Framework 2.8.4.

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.

New in Robot Framework 2.8.

### 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.

Support for modifying tags was added in Robot Framework 2.7.4 and HTML message support in 2.8.

### `fatal_error (msg=None)`

Stops the whole test execution.

The test or suite where this keyword is used fails with the provided message, and subsequent tests fail with a canned message. Possible tear-downs will nevertheless be executed.

See `Fail` if you only want to stop one test case unconditionally.

### `get_count (item1, item2)`

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

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

### `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 added in Robot Framework 2.8.5 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. This time is got using Python’s `time.time()` function.

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

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

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

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

**get_variable_value**(name, default=None)

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

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

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.

Note: Prior to Robot Framework 2.7.4 variables were returned as a custom object that did not support all dictionary methods.

**import_library**(name, *args)

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

This keyword supports importing libraries both using library names and physical paths. When 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)
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.

Logging can be configured using optional html, console and repr arguments. They are off by default, but can be enabled by giving them a true value. See Boolean arguments section for more information about true and false values.
If the `html` argument is given a true value, the message will be considered HTML and special characters such as `<` in it 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.

If the `repr` argument is true, the given item will be passed through a custom version of Python’s `pprint.pformat()` function before logging it. This is useful, for example, when working with strings or bytes containing invisible characters, or when working with nested data structures. The custom version differs from the standard one so that it omits the `u` prefix from Unicode strings and adds `b` prefix to byte strings.

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.

Arguments `html`, `console`, and `repr` are new in Robot Framework 2.8.2.

Pprint support when `repr` is used is new in Robot Framework 2.8.6, and it was changed to drop the `u` prefix and add the `b` prefix in Robot Framework 2.9.

`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.

New in Robot Framework 2.8.2.

`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.

New in Robot Framework 2.8.

```python
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`.

New in Robot Framework 2.8.

```python
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.

```python
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.

```python
remove_tags(*tags)
```

Removes given tags from the current test or all tests in a suite.

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

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

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

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.

```python
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`. Both of these keywords are new in Robot Framework 2.8.

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`.

New in Robot Framework 2.8.

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

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

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.

New in Robot Framework 2.8.2.

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.

New in Robot Framework 2.8.2.

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.
New in Robot Framework 2.7.6.

```
run_keyword_if(condition, name, *args)
```

Runs the given keyword with the given arguments, if `condition` is true.

The given `condition` is evaluated 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*.

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

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

```
The return value is the one of the keyword that was executed or 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 (to conditionally assign fixed values to variables, see *Set Variable If*). 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`.

Starting from Robot Framework 2.8, Python’s [http://docs.python.org/2/library/os.html|os] and [http://docs.python.org/2/library/sys.html|sys] 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.

run_keywords(*keywords)
Executes all the given keywords in a sequence.
This keyword is mainly useful in setups and tear downs when they need to take care 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.
Starting from Robot Framework 2.7.6, keywords can also be run with arguments using upper case AND as a separator between keywords. The keywords are executed so that the first argument is the first keyword and proceeding arguments until the first AND are arguments to it. First argument after the first AND is the second keyword and proceeding arguments until the next AND are its arguments. And so on.
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 test cases and suites executed after setting them.
Setting variables with this keyword thus has the same effect as creating from the command line using the options --variable or --variablefile. Because this keyword can change variables everywhere, it should be used with care.
See Set Suite Variable for more information and examples.

set_library_search_order(*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 $SUITE DOCUMENTATION$.

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

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

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

This keyword sets the metadata of the current suite by default. If the optional top argument is given 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.

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

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

Variables set with this keyword are available everywhere within the scope of the currently executed test suite. Setting variables with this keyword thus has the same effect as creating them using the Variable table in the test data file or importing them from variable files.

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_test_documentation (doc, append=False)
Sets documentation for the current test case.

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

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

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

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

If 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.

Support for append was added in Robot Framework 2.7.7 and support for HTML format in 2.8.

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(*)
```

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, *)
```

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)
```

Fails if the given objects are unequal.

Optional msg and values 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, the error message is simply $<msg>$. See Boolean arguments for more details about using false values.

If ignore_case is given a true value (see Boolean arguments) and arguments are strings, it indicates that comparison should be case-insensitive. New option in Robot Framework 3.0.1.

If both arguments are multiline strings, the comparison is done using multiline string comparisons.

```
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/.

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)

Fails if objects are unequal after converting them to strings.

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

If ignore_case is given a true value (see Boolean arguments), it indicates that comparison should be case-insensitive. New option in Robot Framework 3.0.1.

If both arguments are multiline strings, the comparison is done using multiline string comparisons.

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/2/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.

Starting from Robot Framework 2.8, Should Be True automatically imports Python’s [http://docs.python.org/2/library/os.html] and [http://docs.python.org/2/library/sys.html] 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 unless the given `string` matches the given `pattern`.

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

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

Regular expression check is implemented using the Python [http://docs.python.org/2/library/re.html|re module]. Python’s regular expression syntax is derived from Perl, and it is thus also very similar to the syntax used, for example, in Java, Ruby and .NET.

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

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

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

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

4) Possible flags altering how the expression is parsed (e.g. `re.IGNORECASE`, `re.MULTILINE`) can be set by prefixing the pattern with the `(?<flags>)` group like `(?im)pattern`. The available flags are `i` (case-insensitive), `m` (multiline mode), `s` (dotall mode), `x` (verbose), `u` (Unicode dependent) and `L` (locale dependent).

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, it indicates that comparison should be case-insensitive. 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.
If *ignore_case* is given a true value (see *Boolean arguments*), it indicates that comparison should be case-insensitive. New option in Robot Framework 3.0.1.
See *Should Be Equal* for an explanation on how to override the default error message with *msg* and *values*.

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

**should_not_contain**(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.

```python
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.

```python
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, and it is always case-sensitive. In the pattern `*` matches to anything and `?` matches to any single character.

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

```python
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.

```python
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.

```python
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.

```python
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 5 milliseconds` 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.

```python
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`.

```python
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`. 

4.1. robot package
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. Starting from Robot Framework 2.7, 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](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

```python
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

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 either an empty string or case-insensitively equal to false, none or no. 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/2/library/stdtypes.html#truth-value-testing|rules as in Python].

True examples:

False examples:

Note that prior to Robot Framework 2.9 some keywords considered all non-empty strings, including False, to be true. Considering none false is new in Robot Framework 3.0.3.

= 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 = '3.1.0.dev1'

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

Fails if pattern is not found in list.

See List Should Contain Value for an explanation of msg.

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/2/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.

The given list is never altered by this keyword.

See also Should Not Contain Match.

New in Robot Framework 2.8.6.

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.

New in Robot Framework 2.8.6.

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.
New in Robot Framework 2.8.6.

**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*.

New in Robot Framework 2.8.6.

**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 dictionaries. 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)*  
Returns a copy of the given dictionary.

The given dictionary is never altered by this keyword.

**copy_list** *(list)*  
Returns a copy of the given list.

The given list is never altered by this keyword.

**count_values_in_list** *(list, value, start=0, end=None)*  
Returns the number of occurrences of the given value in list.

The search can be narrowed to the selected sublist by the start and end indexes having the same semantics as 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.

The given dictionaries are never altered by this keyword.

**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.

See Lists Should Be Equal for an explanation of msg. The given dictionary is never altered by this keyword.
**dictionary_should_contain_key** *(dictionary, key, msg=None)*
Fails if key is not found from dictionary.

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

The given dictionary is never altered by this keyword.

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

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

The given dictionaries are never altered by this keyword.

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

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

The given dictionary is never altered by this keyword.

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

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

The given dictionary is never altered by this keyword.

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

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

The given dictionary is never altered by this keyword.

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

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

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

If keys are sortable, they are returned in sorted order. The given dictionary is never altered by this keyword.

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

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

**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.

If the keyword fails, the default error messages is <list> does not contain value '<value>'. A custom message can be given using the msg argument.

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.

See List Should Contain Value for an explanation of msg.

lists_should_be_equal (list1, list2, msg=None, values=True, names=None)

Fails if given lists are unequal.
The keyword first verifies that the lists have equal lengths, and then it checks 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.

```python
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.

```python
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.

```python
pop_from_dictionary (dictionary, *keys)
```
Removes the given keys from the dictionary.
If the given key cannot be found from the dictionary, it is ignored.

```python
remove_duplicates (list,)
```
Returns a list without duplicates based on the given list.
Creates and returns a new list that contains all items in the given list so that one item can appear only once. Order of the items in the new list is the same as in the original except for missing duplicates. Number of the removed duplicates is logged.
New in Robot Framework 2.7.5.

```python
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.

```python
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:

    Starting from Robot Framework 2.8.1, items can also be given as kwargs using key=value syntax:

    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.

    The strings are sorted alphabetically and the numbers numerically.

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

$L = [2,1,’a’,’c’,’b’]$

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.

This library is new in Robot Framework 2.8.5.
= 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 [https://docs.python.org/2/library/datetime.html|datetime] module uses. Basically its [https://docs.python.org/2/library/datetime.html#datetime-objects|datetime] and [https://docs.python.org/2/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 [https://docs.python.org/2/library/datetime.html#strftime-strptime-behavior|datetime.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](http://bugs.jython.org/issue2285)

== Python datetime ==

Python’s standard [https://docs.python.org/2/library/datetime.html#datetime.datetime|datetime.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 [https://docs.python.org/2/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/2/library/stdtypes.html#truth-value-testing|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.DateTimePicker.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.DateTimePicker.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.

```python
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*.

```python
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:

```python
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.

```python
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*.  

---

**4.1. robot package**
• **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.

```python
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*.

```python
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 since Robot Framework 2.8. 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.

```python
robot.libraries.Dialogs.pause_execution(message='Test execution paused. Press OK to continue.')
```

Pauses test execution until user clicks Ok button.

**message** is the message shown in the dialog.

```python
robot.libraries.Dialogs.execute_manual_step(message, default_error='"
```

Pauses test execution until user sets the keyword status.

**User** can 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/2/library/stdtypes.html#truth-value-testing|rules as in Python].

Possibility to hide the typed in value is new in Robot Framework 2.8.4. 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.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 _glob patterns_ where:

Unless otherwise noted, matching is case-insensitive on case-insensitive operating systems such as Windows. Pattern matching is implemented using [http://docs.python.org/library/fnmatch.html|fnmatch module].

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.

Tilde expansion is a new feature in Robot Framework 2.8. 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 either an empty string or case-insensitively equal to `false`, `none` or `no`. Other strings are considered true regardless their value, and other argument types are tested using the same [http://docs.python.org/2/library/stdtypes.html#truth-value-testing|rules as in Python].

True examples:

False examples:

Prior to Robot Framework 2.9, all non-empty strings, including `false` and `no`, were considered true. Considering `none false` is new in Robot Framework 3.0.3.

= Example =

```
ROBOT_LIBRARY_SCOPE = 'GLOBAL'
ROBOT_LIBRARY_VERSION = '3.1.0.dev1'

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 [Process library](http://robotframework.org/robotframework/latest/libraries/Process.html) supports better process configuration and is generally recommended as a replacement for this keyword.

### Run And Return RC

---

**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 [Process library](http://robotframework.org/robotframework/latest/libraries/Process.html) supports better process configuration and is generally recommended as a replacement for this keyword.

### Run And Return RC And Output

---

**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 [Process library](http://robotframework.org/robotframework/latest/libraries/Process.html) supports better process configuration and is generally recommended as a replacement for this keyword.

### Get File

---

**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.

**encoding_errors** argument was added in Robot Framework 2.8.5 and the support for SYSTEM and CONSOLE encodings in Robot Framework 3.0.

### Get Binary File

---

**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

---

**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

---

4.1. robot package

---

59
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.

coding_errors argument is new in Robot Framework 2.8.5.

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.

coding_errors argument is new in Robot Framework 2.8.5.

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.

\texttt{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 \textit{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.

\texttt{directory\_should\_be\_empty\ (path, msg=None)}

Fails unless the specified directory is empty.

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

\texttt{directory\_should\_not\_be\_empty\ (path, msg=None)}

Fails if the specified directory is empty.

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

\texttt{file\_should\_be\_empty\ (path, msg=None)}

Fails unless the specified file is empty.

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

\texttt{file\_should\_not\_be\_empty\ (path, msg=None)}

Fails if the specified directory is empty.

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

\texttt{create\_file\ (path, content='", encoding='UTF-8')}

Creates a file with the given content and encoding.

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

See \textit{Get File} for more information about possible \texttt{encoding} values, including special values \texttt{SYSTEM} and \texttt{CONSOLE}.

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

The support for \texttt{SYSTEM} and \texttt{CONSOLE} encodings is new in Robot Framework 3.0.

\texttt{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.

New in Robot Framework 2.8.5.

**append_to_file** *(path, content, encoding=’UTF-8’)*

Appends the given content to the specified file.

If the file does not exists, this keyword works exactly the same way as *Create File*.

**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 an existing file. Starting from Robot Framework 2.8.4, it can be given as 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.

New in Robot Framework 2.8.4.

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.

New in Robot Framework 2.8.4.

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.

generate_environment_variable(name, default=None)

Returns the value of an environment variable with the given name.

If no such environment variable is set, returns the default value, if given. Otherwise fails the test case.

Starting from Robot Framework 2.7, returned variables are automatically decoded to Unicode using the system encoding.

Note that you can also access environment variables directly using the variable syntax %{ENV_VAR_NAME}.

set_environment_variable(name, value)

Sets an environment variable to a specified value.

Values are converted to strings automatically. Starting from Robot Framework 2.7, set variables are automatically encoded using the system encoding.

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.

New in Robot Framework 2.8.4.

**remove_environment_variable**(*names*)

Deletes the specified environment variable.

Does nothing if the environment variable is not set.

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

**environment_variable_should_be_set**(*name, msg=None*)

Fails if the specified environment variable is not set.

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

**environment_variable_should_not_be_set**(*name, msg=None*)

Fails if the specified environment variable is set.

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

**get_environment_variables**()

Returns currently available environment variables as a dictionary.

Both keys and values are decoded to Unicode using the system encoding. Altering the returned dictionary has no effect on the actual environment variables.

New in Robot Framework 2.7.

**log_environment_variables**(*level='INFO'*)

Logs all environment variables using the given log level.

Environment variables are also returned the same way as with Get Environment Variables keyword.

New in Robot Framework 2.7.

**join_path**(*base, *parts*)

Joins the given path part(s) to the given base path.

The path separator (/ or \) is inserted when needed and the possible absolute paths handled as expected. The resulted path is also normalized.

- ${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)*

Normalizes the given path.

- `${path}` = ‘abc’
- `${p2}` = ‘abc’
- `${p3}` = ‘def’
- `${p4}` = ‘abc/def’
- `${p5}` = ‘abc/def’

**split_path** *(path)*

Splits the given path from the last path separator (/ or \).

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.

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
   177654467 seconds after the epoch.

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

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

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

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

Support for UTC time is a new feature in Robot Framework 2.7.5.

get_file_size (path)
    Returns and logs file size as an integer in bytes.

list_directory (path, pattern=None, absolute=False)
    Returns and logs items in a directory, optionally filtered with pattern.

File and directory names are returned in case-sensitive alphabetical order, e.g. ['A Name',
'Second', 'a lower case name', 'one more']]. Implicit directories . and .. are not re-
turned. 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**

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

Robot Framework test library for running processes.


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*.

This library is new in Robot Framework 2.8.
```

== 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.

Starting from Robot Framework 2.8.6, 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 either an empty string or case-insensitively equal to `false`, `none` or `no`. Other strings are considered true regardless their value, and other argument types are tested using the same [rules as in Python](http://docs.python.org/2/library/stdtypes.html#truth-value-testingrules as in Python).

True examples:

False examples:

Prior to Robot Framework 2.9, all non-empty strings, including `false` and `no`, were considered to be true. Considering `none` false is new in Robot Framework 3.0.3.

= Example =

```python
ROBOT_LIBRARY_SCOPE = 'GLOBAL'
ROBOT_LIBRARY_VERSION = '3.1.0.dev1'
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.

timeout and on_timeout arguments are new in Robot Framework 2.8.4.

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.

  Starting from Robot Framework 2.8.5, 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 by Jython in general nor by Python versions prior to 2.7 on Windows.

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.

timeout and on_timeout are new in Robot Framework 2.8.2.

**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 by Jython nor by Python versions prior to 2.7. Process is killed instead.

• Stopping the whole process group is not supported by Jython at all nor by Python versions prior to 2.7 on Windows.

• On Windows forceful kill only stops the main process, not possible child processes.

Automatically killing the process if termination fails as well as returning a result object are new features in Robot Framework 2.8.2. Terminating also possible child processes, including using CTRL_BREAK_EVENT on Windows, is new in Robot Framework 2.8.5.

**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.

New in Robot Framework 2.8.2. Support for `group` argument is new in Robot Framework 2.8.5.

**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.

New in Robot Framework 2.8.2.

**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.
    
    Support for timeouts is a new feature in Robot Framework 2.8.6. Timeouts do not work with IronPython.
    
    ROBOT_LIBRARY_SCOPE = 'TEST SUITE'
    get_keyword_names(attempts=2)
    get_keyword_arguments (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_tags(name)
get_keyword_documentation(name)
run_keyword(name, args, kwargs)

class robot.libraries.Remote.TimeoutTransport (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)'

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](http://wxpython.org) :: Required also by RIDE so many Robot Framework users already have this module installed.
- **PyGTK** :: [http://pygtk.org](http://pygtk.org) :: This module is available by default on most Linux distributions.
- **Pillow** :: [https://python-pillow.github.io](https://python-pillow.github.io) :: Only works on Windows. Also the original PIL package is supported.
- **Scrot** :: [https://en.wikipedia.org/wiki/Scrot](https://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
ROBOT_LIBRARY_SCOPE = 'TEST SUITE'
ROBOT_LIBRARY_VERSION = '3.1.0.dev1'

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 = '3.1.0.dev1'

class convert_to_lowercase (string)

Converts string to lowercase.

New in Robot Framework 2.8.6.

class convert_to_uppercase (string)

Converts string to uppercase.

New in Robot Framework 2.8.6.
**encode_string_to_bytes** *(string, encoding, errors='strict')*

Encodes the given Unicode string to bytes using the given encoding.

The `errors` argument controls what to do if encoding some characters fails. All values accepted by the `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.

The `errors` argument controls what to do if decoding some bytes fails. All values accepted by the `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.

**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 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 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 concatenated 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.

New in Robot Framework 2.8.2.

**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.

New in Robot Framework 2.8.2.

**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 the given string is not a 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.
= 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

Using UTF-8 encoding by default and being able to configure the encoding are new features in Robot Framework 2.7.6. In earlier versions only ASCII was supported and encoding errors were silently ignored. Robot Framework 2.7.7 added a possibility to specify the error handler, changed the default behavior back to ignoring encoding errors, and added the possibility to disable encoding.

== Default log level ==

Default log level specifies the log level keywords use for logging unless they are given an explicit log level. The default value is INFO, and changing it, for example, to DEBUG can be a good idea if there is lot of unnecessary output that makes log files big.

Configuring default log level in importing and with Open Connection are new features in Robot Framework 2.7.6. In earlier versions only Set Default Log Level could be used.

== 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.

New in Robot Framework 2.8.2.

== Window size ==

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

New in Robot Framework 2.8.2.
== 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`.

New in Robot Framework 2.8.2.

= Terminal emulation =

Starting from Robot Framework 2.8.2, Telnet library supports terminal emulation with [https://pyte.readthedocs.io](https://pyte.readthedocs.io). 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 [Configurationconfigured] default value.

The valid log levels to use are `TRACE`, `DEBUG`, `INFO` (default), and `WARN`. Levels below `INFO` are not shown in log files by default whereas warnings are shown more prominently.

The [http://docs.python.org/2/library/telnetlib.html#telnetlib module] used by this library has a custom logging system for logging content it sends and receives. By default these messages are written using `TRACE` level.

Starting with Robot Framework 2.8.7 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` can 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](http://robotframework.org/robotframework/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 either an empty string or case-insensitively equal to `false`, `none` or `no`. Other strings are considered true regardless their value, and other argument types are tested using the same [http://docs.python.org/2/library/stdtypes.html#truth-value-testing rules as in Python].
True examples:

False examples:

Prior to Robot Framework 2.9, all non-empty strings, including `false` and `no`, were considered to be true. Considering `none` false is new in Robot Framework 3.0.3.

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.

```python
ROBOT_LIBRARY_SCOPE = 'TEST_SUITE'
ROBOT_LIBRARY_VERSION = '3.1.0.dev1'

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 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.

```python
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.

```python
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/2/library/re.htmlPython re module] for more information about the supported regular expression syntax. Notice that possible backslashes need to be escaped in Robot Framework test data.

See Configuration section for more information about global and connection specific configuration.

set_encoding (encoding=None, errors=None)
Sets the encoding to use for writing and reading in the current connection.

The given encoding specifies the encoding to use when written/read text is encoded/decoded, and errors specifies the error handler to use if encoding/decoding fails. Either of these can be omitted and in that case the old value is not affected. Use string NONE to disable encoding altogether.
See Configuration section for more information about encoding and error handlers, as well as global and connection specific configuration in general.

The old values are returned and can be used to restore the encoding and the error handler later. See Set Prompt for a similar example.

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

Setting encoding in general is a new feature in Robot Framework 2.7.6. Specifying the error handler and disabling encoding were added in 2.7.7.

**set_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.

New in Robot Framework 2.8.7.

**set_default_log_level** (level)
Sets the default log level used for logging in the current connection.

The old default log level is returned and can be used to restore the log level later.

See Configuration section for more information about global and connection specific configuration.

**close_connection** (loglevel=None)
Closes the current Telnet connection.

Remaining output in the connection is read, logged, and returned. It is not an error to close an already closed connection.

Use Close All Connections if you want to make sure all opened connections are closed.

See Logging section for more information about log levels.

**login** (username, password, login_prompt='login: ', password_prompt='Password: ', login_timeout='1 second', login_incorrect='Login incorrect')
Logs in to the Telnet server with the given user information.

This keyword reads from the connection until the login_prompt is encountered and then types the given username. Then it reads until the password_prompt and types the given password. In both cases a newline is appended automatically and the connection specific timeout used when waiting for outputs.

How logging status is verified depends on whether a prompt is set for this connection or not:

1) If the prompt is set, this keyword reads the output until the prompt is found using the normal timeout. If no prompt is found, login is considered failed and also this keyword fails. Note that in this case both login_timeout and login_incorrect arguments are ignored.

2) If the prompt is not set, this keywords sleeps until login_timeout and then reads all the output available on the connection. If the output contains login_incorrect text, login is considered failed and also this keyword fails. Both of these configuration parameters were added in Robot Framework 2.7.6.

In earlier versions they were hard coded.

See Configuration section for more information about setting newline, timeout, and prompt.

**write** (text, loglevel=None)
Writes the given text plus a newline into the connection.

The newline character sequence to use can be [#Configuration|configured] both globally and per connection basis. The default value is CRLF.
This keyword consumes the written text, until the added newline, from the output and logs and returns it. The given text itself must not contain newlines. Use Write Bare instead if either of these features causes a problem.

Note: This keyword does not return the possible output of the executed command. To get the output, one of the Read ... keywords must be used. See Writing and reading section for more details.

See Logging section for more information about log levels.

write_bare (text)
Writes the given text, and nothing else, into the connection.

This keyword does not append a newline nor consume the written text. Use Write if these features are needed.

write_until_expected_output (text, expected, timeout, retry_interval, loglevel=None)
Writes the given text repeatedly, until expected appears in the output.

text is written without appending a newline and it is consumed from the output before trying to find expected. If expected does not appear in the output within timeout, this keyword fails.

retry_interval defines the time to wait expected to appear before writing the text again. Consuming the written text is subject to the normal [#Configuration|configured timeout].

Both timeout and retry interval must be given in time string format. See Logging section for more information about log levels.

The above example writes command ps -ef | grep myprocess until myprocess appears in the output. The command is written every 0.5 seconds and the keyword fails if myprocess does not appear in the output in 5 seconds.

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/2/library/re.html|Python re module] for more information about the supported regular expression syntax. Notice that possible backslashes need to be escaped in Robot Framework test data.

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.

Optionally stripping prompt is a new feature in Robot Framework 2.8.7.

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()
Interation 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.

**sockavail()**

Test whether data is available on the socket.

```python
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**

```python
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 [https://docs.python.org/2/library/xml.etree.elementtree.html|ElementTree XML API].

The library has the following main usages:

- Parsing an XML file, or a string containing XML, into an XML element structure and finding certain elements from it for further analysis (e.g. Parse XML and Get Element keywords).
- Getting text or attributes of elements (e.g. Get Element Text and Get Element Attribute).
- Directly verifying text, attributes, or whole elements (e.g. Element Text Should Be and Elements Should Be Equal).
- Modifying XML and saving it (e.g. Set Element Text, Add Element and Save XML).
```

== Table of contents ==

- Parsing XML
- Using lxml
- Example
- Finding elements with xpath
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. With IronPython parsing XML with a doctype is not supported at all.

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 [ElementTree](https://docs.python.org/2/library/xml.etree.elementtree.html) module for parsing XML, but it can be configured to use 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.

The lxml support is new in Robot Framework 2.8.5.

### 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.
ElementTree, and thus also this library, supports finding elements using xpath expressions. ElementTree does not, however, support the full xpath syntax, and what is supported depends on its version. ElementTree 1.3 that is distributed with Python 2.7 supports richer syntax than earlier versions.

The supported xpath syntax is explained below and [http://effbot.org/zone/element-xpath.html|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. This syntax is supported only in ElementTree 1.3 (i.e. Python/Jython 2.7 and newer).

== 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 above.

What predicates ElementTree supports is explained in the table below. Notice that predicates in general are supported only in ElementTree 1.3 (i.e. Python/Jython 2.7 and newer).

Predicates can also be stacked like path[predicate1][predicate2]. A limitation is that possible position predicate must always be first.

= Element attributes =

All keywords returning elements, such as Parse XML, and Get Element, return ElementTree’s [http://docs.python.org/library/xml.etree.elementtree.html#xml.etree.ElementTree.Element|Element objects]. These elements can be used as inputs for other keywords, but they also contain several useful attributes that can be accessed directly using the extended variable syntax.

The attributes that are both useful and convenient to use in the test data are explained below. Also other attributes, including methods, can be accessed, but that is typically better to do in custom libraries than directly in the test data.

The examples use the same $\{XML\}$ structure as the earlier examples.
== tag ==
The tag of the element.

== 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 either an empty string or case-insensitively equal to `false`, `none` or `no`. Other strings are considered true regardless their value, and other argument types are tested using the same [truth-value-testing](http://docs.python.org/2/library/stdtypes.html) as in Python.

**True examples:**

**False examples:**

Prior to Robot Framework 2.9, all non-empty strings, including `false` and `no`, were considered to be true. Considering `none` false is new in Robot Framework 3.0.3.

Import library with optionally lxml mode enabled.

By default this library uses Python’s standard [ElementTree](https://docs.python.org/2/library/xml.etree.elementtree.html#xml.etree.ElementTree) module for parsing XML. If `use_lxml` argument is given a true value (see [Boolean arguments](#)), the library will use [lxml](http://lxml.de) 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.

The support for lxml is new in Robot Framework 2.8.5.

**ROBOT_LIBRARY_SCOPE = 'GLOBAL'**

**ROBOT_LIBRARY_VERSION = '3.1.0.dev1'**

**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 [ElementTree](https://docs.python.org/2/library/xml.etree.elementtree.html#xml.etree.ElementTree) 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](#)).

4.1. robot package 95
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.

```python
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.

```python
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.

```python
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.

```python
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`.

```python
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.

```python
element_should_not_exist (source, xpath='.', message=None)
```

Verifies that no element match the given `xpath`. 

96 Chapter 4. All packages
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.

```python
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.

```python
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.

```python
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.

```python
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, and it is always case-sensitive. In the pattern, ‘*’ matches anything and ‘?’ matches any single character.

```python
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, and it is always case-sensitive. In the pattern, ‘*’ matches anything and ‘?’ matches any single character.

`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, and it is always case-sensitive. In the pattern, ‘*’
matches anything and ‘?’ matches any single character.

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 se-
mandics 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.

New in Robot Framework 2.8.6.

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 se-
mantics 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.

New in Robot Framework 2.8.6.

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`.

New in Robot Framework 2.8.6.

**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`.

New in Robot Framework 2.8.6.

**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`.

New in Robot Framework 2.8.6.

**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.

```robot
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.

```robot
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`.

```robot
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.

```robot
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.

```robot
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.

```robot
log_element(source, level='INFO', xpath=')
```
Logs the string representation of the specified element.
The element specified with `source` and `xpath` is first converted into a string using `Element To String` keyword internally. The resulting string is then logged using the given `level`. The logged string is also returned.

```python
save_xml(source, path, encoding='UTF-8')
```

Saves the given element to the specified file.

The element to save is specified with `source` using the same semantics as with `Get Element` keyword. The file where the element is saved is denoted with `path` and the encoding to use with `encoding`. The resulting file 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.

```python
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. New in Robot Framework 2.8.5.

```python
class robot.libraries.XML.NameSpaceStripper(etree, lxml_etree=False)
```

```python
strip(elem, preserve=True, current_ns=None, top=True)
unstrip(elem, current_ns=None, copied=False)
```

```python
class robot.libraries.XML.ElementFinder(etree, modern=True, lxml=False)
```

```python
find_all(elem, xpath)
```

```python
class robot.libraries.XML.ElementComparator(comparator, normalizer=None, exclude_children=False)
```

```python
compare(actual, expected, location=None)
```

```python
class robot.libraries.XML.Location(path, is_root=True)
```

```python
child(tag)
```
robot.libraries.dialogs_ipy module

robot.libraries.dialogs_jy module

class robot.libraries.dialogs_jy.MessageDialog(message)
    Bases: robot.libraries.dialogs_jy._SwingDialog
    show()

class robot.libraries.dialogs_jy.InputDialog(message, default, hidden=False)
    Bases: robot.libraries.dialogs_jy._SwingDialog
    show()

class robot.libraries.dialogs_jy.SelectionDialog(message, options)
    Bases: robot.libraries.dialogs_jy._SwingDialog
    show()

class robot.libraries.dialogs_jy.PassFailDialog(message)
    Bases: robot.libraries.dialogs_jy._SwingDialog
    show()

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, XXXX
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 appli-
cation. Return current command if VALUE is None.

config(cnf=None, **kw)

Configure resources of a widget.

The values for resources are specified as keyword arguments. To get an overview about the allowed
keyword arguments call the method keys.

configure(cnf=None, **kw)

Configure resources of a widget.

The values for resources are specified as keyword arguments. To get an overview about the allowed
keyword arguments call the method keys.

deiconify()

Deiconify this widget. If it was never mapped it will not be mapped. On Windows it will raise this widget
and give it the focus.

deletecommand(name)

Internal function.

Delete the Tcl command provided in NAME.
**destroy()**

Destroy this and all descendants widgets.

**event_add**(virtual, *sequences)

Bind a virtual event VIRTUAL (of the form <<Name>>) to an event SEQUENCE such that the virtual event is triggered whenever SEQUENCE occurs.

**event_delete**(virtual, *sequences)

Unbind a virtual event VIRTUAL from SEQUENCE.

**event_generate**(sequence, **kw)

Generate an event SEQUENCE. Additional keyword arguments specify parameter of the event (e.g. x, y, rootx, rooty).

**event_info**(virtual=None)

Return a list of all virtual events or the information about the SEQUENCE bound to the virtual event VIRTUAL.

**focus()**

Direct input focus to this widget.

If the application currently does not have the focus this widget will get the focus if the application gets the focus through the window manager.

**focus_displayof()**

Return the widget which has currently the focus on the display where this widget is located.

Return None if the application does not have the focus.

**focus_force()**

Direct input focus to this widget even if the application does not have the focus. Use with caution!

**focus_get()**

Return the widget which has currently the focus in the application.

Use focus_displayof to allow working with several displays. Return None if application does not have the focus.

**focus_lastfor()**

Return the widget which would have the focus if top level for this widget gets the focus from the window manager.

**focus_set()**

Direct input focus to this widget.

If the application currently does not have the focus this widget will get the focus if the application gets the focus through the window manager.

**focusmodel**(model=None)

Set focus model to MODEL. “active” means that this widget will claim the focus itself, “passive” means that the window manager shall give the focus. Return current focus model if MODEL is None.

**frame()**

Return identifier for decorative frame of this widget if present.

**geometry**(newGeometry=None)

Set geometry to NEWGEOMETRY of the form =widthxheight+x+y. Return current value if None is given.

**getboolean**(s)

Return a boolean value for Tcl boolean values true and false given as parameter.

**getdouble**

alias of float
getint
   alias of int

getvar (name='PY_VAR')
   Return value of Tcl variable NAME.

grab_current ()
   Return widget which has currently the grab in this application or None.

grab_release ()
   Release grab for this widget if currently set.

grab_set (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 descen-
dents that don’t have an icon set explicitly. DEFAULT can be the relative path to a .ico file (example:root.iconbitmap(default='myicon.ico') ). See Tk documentation for more information.

iconify ()
    Display widget as icon.

iconmask (bitmap=None)
    Set mask for the icon bitmap of this widget. Return the mask if None is given.

iconname (newName=None)
    Set the name of the icon for this widget. Return the name if None is given.

iconposition (x=None, y=None)
    Set the position of the icon of this widget to X and Y. Return a tuple of the current values of X and X if
    None is given.

iconwindow (pathName=None)
    Set widget PATHNAME to be displayed instead of icon. Return the current value if None is given.

image_names ()
    Return a list of all existing image names.

image_types ()
    Return a list of all available image types (e.g. photo bitmap).

keys ()
    Return a list of all resource names of this widget.

lift (aboveThis=None)
    Raise this widget in the stacking order.

lower (belowThis=None)
    Lower this widget in the stacking order.

mainloop (n=0)
    Call the mainloop of Tk.

maxsize (width=None, height=None)
    Set max WIDTH and HEIGHT for this widget. If the window is gridded the values are given in grid units.
    Return the current values if None is given.

minsize (width=None, height=None)
    Set min WIDTH and HEIGHT for this widget. If the window is gridded the values are given in grid units.
    Return the current values if None is given.

nametowidget (name)
    Return the Tkinter instance of a widget identified by its Tcl name NAME.

option_add (pattern, value, priority=None)
    Set a VALUE (second parameter) for an option PATTERN (first parameter).
    An optional third parameter gives the numeric priority (defaults to 80).
option_clear ()
  Clear the option database.
  It will be reloaded if option_add is called.

option_get (name, className)
  Return the value for an option NAME for this widget with CLASSNAME.
  Values with higher priority override lower values.

option_readfile (fileName, priority=None)
  Read file FILENAME into the option database.
  An optional second parameter gives the numeric priority.

overriderdirect (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.

\texttt{winfo\_atom(name, displayof=0)}
\begin{itemize}
  \item Return integer which represents atom \textit{Name}.
\end{itemize}

\texttt{winfo\_atominame(id, displayof=0)}
\begin{itemize}
  \item Return name of atom with identifier \textit{ID}.
\end{itemize}

\texttt{winfo\_cells()}
\begin{itemize}
  \item Return number of cells in the colormap for this widget.
\end{itemize}

\texttt{winfo\_children()}
\begin{itemize}
  \item Return a list of all widgets which are children of this widget.
\end{itemize}

\texttt{winfo\_class()}
\begin{itemize}
  \item Return window class name of this widget.
\end{itemize}

\texttt{winfo\_colormapfull()}
\begin{itemize}
  \item Return true if at the last color request the colormap was full.
\end{itemize}

\texttt{winfo\_containing(rootX, rootY, displayof=0)}
\begin{itemize}
  \item Return the widget which is at the root coordinates \textit{ROOTX}, \textit{ROOTY}.
\end{itemize}

\texttt{winfo\_depth()}
\begin{itemize}
  \item Return the number of bits per pixel.
\end{itemize}

\texttt{winfo\_exists()}
\begin{itemize}
  \item Return true if this widget exists.
\end{itemize}

\texttt{winfo\_fpixels(number)}
\begin{itemize}
  \item Return the number of pixels for the given distance \textit{NUMBER} (e.g. “3c”) as float.
\end{itemize}

\texttt{winfo\_geometry()}
\begin{itemize}
  \item Return geometry string for this widget in the form “widthxheight+X+Y”.
\end{itemize}

\texttt{winfo\_height()}
\begin{itemize}
  \item Return height of this widget.
\end{itemize}

\texttt{winfo\_id()}
\begin{itemize}
  \item Return identifier \textit{ID} for this widget.
\end{itemize}

\texttt{winfo\_interps(displayof=0)}
\begin{itemize}
  \item Return the name of all Tcl interpreters for this display.
\end{itemize}

\texttt{winfo\_ismapped()}
\begin{itemize}
  \item Return true if this widget is mapped.
\end{itemize}

\texttt{winfo\_manager()}
\begin{itemize}
  \item Return the window manager name for this widget.
\end{itemize}

\texttt{winfo\_name()}
\begin{itemize}
  \item Return the name of this widget.
\end{itemize}

\texttt{winfo\_parent()}
\begin{itemize}
  \item Return the name of the parent of this widget.
\end{itemize}

\texttt{winfo\_pathname(id, displayof=0)}
\begin{itemize}
  \item Return the pathname of the widget given by \textit{ID}.
\end{itemize}

\texttt{winfo\_pixels(number)}
\begin{itemize}
  \item Rounded integer value of \texttt{winfo\_fpixels}.
\end{itemize}

\texttt{winfo\_pointerx()}
\begin{itemize}
  \item Return the \textit{x} coordinate of the pointer on the root window.
\end{itemize}
\texttt{winfo_pointerxy}()
Return a tuple of x and y coordinates of the pointer on the root window.

\texttt{winfo_pointery}()
Return the y coordinate of the pointer on the root window.

\texttt{winfo_reqheight}()
Return requested height of this widget.

\texttt{winfo_reqwidth}()
Return requested width of this widget.

\texttt{winfo_rgb}(\texttt{color})
Return tuple of decimal values for red, green, blue for COLOR in this widget.

\texttt{winfo_rootx}()
Return x coordinate of upper left corner of this widget on the root window.

\texttt{winfo_rooty}()
Return y coordinate of upper left corner of this widget on the root window.

\texttt{winfo_screen}()
Return the screen name of this widget.

\texttt{winfo_screencells}()
Return the number of the cells in the colormap of the screen of this widget.

\texttt{winfo_screendepth}()
Return the number of bits per pixel of the root window of the screen of this widget.

\texttt{winfo_screenheight}()
Return the number of pixels of the height of the screen of this widget in pixel.

\texttt{winfo_screenmmheight}()
Return the number of pixels of the height of the screen of this widget in mm.

\texttt{winfo_screenmmwidth}()
Return the number of pixels of the width of the screen of this widget in mm.

\texttt{winfo_screenvisual}()
Return one of the strings directcolor, grayscale, pseudocolor, staticcolor, staticgray, or truecolor for the default colormodel of this screen.

\texttt{winfo_screenwidth}()
Return the number of pixels of the width of the screen of this widget in pixel.

\texttt{winfo_server}()
Return information of the X-Server of the screen of this widget in the form \texttt{"XmajorRminor vendor vendorVersion"}.

\texttt{winfo_toplevel}()
Return the toplevel widget of this widget.

\texttt{winfo_viewable}()
Return true if the widget and all its higher ancestors are mapped.

\texttt{winfo_visual}()
Return one of the strings directcolor, grayscale, pseudocolor, staticcolor, staticgray, or truecolor for the colormodel of this widget.

\texttt{winfo_visualid}()
Return the X identifier for the visual for this widget.
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 MINNUMBER/MINDENOM and MAXNUMBER/MAXDENOM. Return a tuple of the actual values if no argument is given.

wm_attributes (*args)
This subcommand returns or sets platform specific attributes

The first form returns a list of the platform specific flags and their values. The second form returns the value for the specific option. The third form sets one or more of the values. The values are as follows:

On Windows, -disabled gets or sets whether the window is in a disabled state. -toolwindow gets or sets the style of the window to toolwindow (as defined in the MSDN). -topmost gets or sets whether this is a topmost window (displays above all other windows).

On Macintosh, XXXXX

On Unix, there are currently no special attribute values.

wm_client (name=None)
Store NAME in WM_CLIENT_MACHINE property of this widget. Return current value.

wm_colormapwindows (*wlist)
Store list of window names (WLIST) into WM_COLORMAPWINDOWS property of this widget. This list contains windows whose colormaps differ from their parents. Return current list of widgets if WLIST is empty.

wm_command (value=None)
Store VALUE in WM_COMMAND property. It is the command which shall be used to invoke the application. Return current command if VALUE is None.
wm_deiconify()
Deiconify this widget. If it was never mapped it will not be mapped. On Windows it will raise this widget and give it the focus.

wm_focusmodel (model=None)
Set focus model to MODEL. “active” means that this widget will claim the focus itself, “passive” means that the window manager shall give the focus. Return current focus model if MODEL is None.

wm_frame()
Return identifier for decorative frame of this widget if present.

wm_geometry (newGeometry=None)
Set geometry to NEWGEOMETRY of the form =widthxheight+x+y. Return current value if None is given.

wm_grid (baseWidth=None, baseHeight=None, widthInc=None, heightInc=None)
Instruct the window manager that this widget shall only be resized on grid boundaries. WIDTHINC and HEIGHTINC are the width and height of a grid unit in pixels. BASEWIDTH and BASEHEIGHT are the number of grid units requested in Tk_GeometryRequest.

wm_group (pathName=None)
Set the group leader widgets for related widgets to PATHNAME. Return the group leader of this widget if None is given.

wm_iconbitmap (bitmap=None, default=None)
Set bitmap for the iconified widget to BITMAP. Return the bitmap if None is given.
Under Windows, the DEFAULT parameter can be used to set the icon for the widget and any descendants that don’t have an icon set explicitly. DEFAULT can be the relative path to a .ico file (example: root.iconbitmap(default='myicon.ico')). See Tk documentation for more information.

wm_iconify()
Display widget as icon.

wm_iconmask (bitmap=None)
Set mask for the icon bitmap of this widget. Return the mask if None is given.

wm_iconname (newName=None)
Set the name of the icon for this widget. Return the name if None is given.

wm_iconposition (x=None, y=None)
Set the position of the icon of this widget to X and Y. Return a tuple of the current values of X and Y if None is given.

wm_iconwindow (pathName=None)
Set widget PATHNAME to be displayed instead of icon. Return the current value if None is given.

wm_maxsize (width=None, height=None)
Set max WIDTH and HEIGHT for this widget. If the window is gridded the values are given in grid units. Return the current values if None is given.

wm_minsize (width=None, height=None)
Set min WIDTH and HEIGHT for this widget. If the window is gridded the values are given in grid units. Return the current values if None is given.

wm_overrideredirect (boolean=None)
Instruct the window manager to ignore this widget if BOOLEAN is given with 1. Return the current value if None is given.

wm_positionfrom (who=None)
Instruct the window manager that the position of this widget shall be defined by the user if WHO is “user”, and by its own policy if WHO is “program”.

4.1. robot package 115
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.
Identifer 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.

gameometry(newGeometry=None)
    Set geometry to NEWGEOMETRY of the form =widthxheight+x+y. Return current value if None is given.

getboolean(s)
    Return a boolean value for Tcl boolean values true and false given as parameter.

gedouble
    alias of float
getint
    alias of int

getvar (name='PY_VAR')
    Return value of Tcl variable NAME.

grab_current ()
    Return widget which has currently the grab in this application or None.

grab_release ()
    Release grab for this widget if currently set.

grab_set (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 \( (row=\text{None}, column=\text{None}) \)
Return a list of all slaves of this widget in its packing order.

group \( (\text{pathName}=\text{None}) \)
Set the group leader widgets for related widgets to \text{PATHNAME}. Return the group leader of this widget if None is given.

iconbitmap \( (\text{bitmap}=\text{None}, \text{default}=\text{None}) \)
Set bitmap for the iconified widget to \text{BITMAP}. Return the bitmap if None is given.

Under Windows, the \text{DEFAULT} parameter can be used to set the icon for the widget and any descendants that don't have an icon set explicitly. \text{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 \( (\text{bitmap}=\text{None}) \)
Set mask for the icon bitmap of this widget. Return the mask if None is given.

iconname \( (\text{newName}=\text{None}) \)
Set the name of the icon for this widget. Return the name if None is given.

iconposition \( (x=\text{None}, y=\text{None}) \)
Set the position of the icon of this widget to \text{X} and \text{Y}. Return a tuple of the current values of \text{X} and \text{X} if None is given.

iconwindow \( (\text{pathName}=\text{None}) \)
Set widget \text{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 \( (\text{aboveThis}=\text{None}) \)
Raise this widget in the stacking order.

lower \( (\text{belowThis}=\text{None}) \)
Lower this widget in the stacking order.

mainloop \( (n=0) \)
Call the mainloop of Tk.

maxsize \( (\text{width}=\text{None}, \text{height}=\text{None}) \)
Set max \text{WIDTH} and \text{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 \( (\text{width}=\text{None}, \text{height}=\text{None}) \)
Set min \text{WIDTH} and \text{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 \( (\text{name}) \)
Return the Tkinter instance of a widget identified by its Tcl name \text{NAME}.

option_add \( (\text{pattern}, \text{value}, \text{priority}=\text{None}) \)
Set a \text{VALUE} (second parameter) for an option \text{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.

overrideset(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.

```python
tk_focusPrev()
```
Return previous widget in the focus order. See tk_focusNext for details.

```python
tk_menuBar(*args)
```
Do not use. Needed in Tk 3.6 and earlier.

```python
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.

```python
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.

```python
tkraise(aboveThis=None)
```
Raise this widget in the stacking order.

```python
transient(master=None)
```
Instruct the window manager that this widget is transient with regard to widget MASTER.

```python
unbind(sequence, funcid=None)
```
Unbind for this widget for event SEQUENCE the function identified with FUNCID.

```python
unbind_all(sequence)
```
Unbind for all widgets for event SEQUENCE all functions.

```python
unbind_class(className, sequence)
```
Unbind for all widgets with bindtag CLASSNAME for event SEQUENCE all functions.

```python
update()
```
Enter event loop until all pending events have been processed by Tcl.

```python
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.

```python
wait_variable(name='PY_VAR')
```
Wait until the variable is modified.

A parameter of type IntVar, StringVar, DoubleVar or BooleanVar must be given.

```python
wait_visibility(window=None)
```
Wait until the visibility of a WIDGET changes (e.g. it appears).

If no parameter is given self is used.

```python
wait_window(window=None)
```
Wait until a WIDGET is destroyed.

If no parameter is given self is used.

```python
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 pixels. If there is no virtual root window return the width of the screen.

winfo_vrootx()
Return the x offset of the virtual root relative to the root window of the screen of this widget.

winfo_vrooty()
Return the y offset of the virtual root relative to the root window of the screen of this widget.

winfo_width()
Return the width of this widget.

winfo_x()
Return the x coordinate of the upper left corner of this widget in the parent.

winfo_y()
Return the y coordinate of the upper left corner of this widget in the parent.

withdraw()
Withdraw this widget from the screen such that it is unmapped and forgotten by the window manager. Re-draw it with wm_deiconify.

wm_aspect (minNumer=None, minDenom=None, maxNumer=None, maxDenom=None)
Instruct the window manager to set the aspect ratio (width/height) of this widget to be between MINNUMER/MINDENOM and MAXNUMER/MAXDENOM. Return a tuple of the actual values if no argument is given.

wm_attributes (*args)
This subcommand returns or sets platform specific attributes.

The first form returns a list of the platform specific flags and their values. The second form returns the value for the specific option. The third form sets one or more of the values. The values are as follows:

On Windows, -disabled gets or sets whether the window is in a disabled state. -toolwindow gets or sets the style of the window to toolwindow (as defined in the MSDN). -topmost gets or sets whether this is a topmost window (displays above all other windows).

On Macintosh, XXXXX

On Unix, there are currently no special attribute values.

wm_client (name=None)
Store NAME in WM_CLIENT_MACHINE property of this widget. Return current value.

wm_colormapwindows (*wlist)
Store list of window names (WLIST) into WM_COLORMAPWINDOWS property of this widget. This list contains windows whose colormaps differ from their parents. Return current list of widgets if WLIST is empty.

wm_command (value=None)
Store VALUE in WM_COMMAND property. It is the command which shall be used to invoke the application. Return current command if VALUE is None.
wm_deiconify()
Deiconify this widget. If it was never mapped it will not be mapped. On Windows it will raise this widget
and give it the focus.

wm_focusmodel (model=None)
Set focus model to MODEL. “active” means that this widget will claim the focus itself, “passive” means
that the window manager shall give the focus. Return current focus model if MODEL is None.

wm_frame()
Return identifier for decorative frame of this widget if present.

wm_geometry (newGeometry=None)
Set geometry to NEWGEOMETRY of the form =widthxheight+x+y. Return current value if None is given.

wm_grid (baseWidth=None, baseHeight=None, widthInc=None, heightInc=None)
Instruct the window manager that this widget shall only be resized on grid boundaries. WIDTHINC and
HEIGHTINC are the width and height of a grid unit in pixels. BASEWIDTH and BASEHEIGHT are the
number of grid units requested in Tk_GeometryRequest.

wm_group (pathName=None)
Set the group leader widgets for related widgets to PATHNAME. Return the group leader of this widget if
None is given.

wm_iconbitmap (bitmap=None, default=None)
Set bitmap for the iconified widget to BITMAP. Return the bitmap if None is given.
Under Windows, the DEFAULT parameter can be used to set the icon for the widget and any descen-
dents that don’t have an icon set explicitly. DEFAULT can be the relative path to a .ico file (example:
root.iconbitmap(default='myicon.ico') ). See Tk documentation for more information.

wm_iconify()
Display widget as icon.

wm_iconmask (bitmap=None)
Set mask for the icon bitmap of this widget. Return the mask if None is given.

wm_iconname (newName=None)
Set the name of the icon for this widget. Return the name if None is given.

wm_iconposition (x=None, y=None)
Set the position of the icon of this widget to X and Y. Return a tuple of the current values of X and X if
None is given.

wm_iconwindow (pathName=None)
Set widget PATHNAME to be displayed instead of icon. Return the current value if None is given.

wm_maxsize (width=None, height=None)
Set max WIDTH and HEIGHT for this widget. If the window is gridded the values are given in grid units.
Return the current values if None is given.

wm_minsize (width=None, height=None)
Set min WIDTH and HEIGHT for this widget. If the window is gridded the values are given in grid units.
Return the current values if None is given.

wm_overrideredirect (boolean=None)
Instruct the window manager to ignore this widget if BOOLEAN is given with 1. Return the current value
if None is given.

wm_positionfrom (who=None)
Instruct the window manager that the position of this widget shall be defined by the user if WHO is “user”,
and by its own policy if WHO is “program”.
```python
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

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, 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 <<AS-tring>> 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**(*wl*)

Store list of window names (WLIST) into WM_COLORMAPWINDOWS property of this widget. This list contains windows whose colormaps differ from their parents. Return current list of widgets if WLIST is empty.

**colormodel**(*value=None*)

Useless. Not implemented in Tk.

**columnconfigure**(*index*, cnf={}, **kw*)

Configure column *INDEX* of a grid.

Valid resources are minsize (minimum size of the column), weight (how much does additional space propagate to this column) and pad (how much space to let additionally).

**command**(*value=None*)

Store VALUE in WM_COMMAND property. It is the command which shall be used to invoke the application. Return current command if VALUE is None.

**config**(*cnf=None*, **kw*)

Configure resources of a widget.

The values for resources are specified as keyword arguments. To get an overview about the allowed keyword arguments call the method keys.

**configure**(*cnf=None*, **kw*)

Configure resources of a widget.

The values for resources are specified as keyword arguments. To get an overview about the allowed keyword arguments call the method keys.

**deiconify**()

Deiconify this widget. If it was never mapped it will not be mapped. On Windows it will raise this widget and give it the focus.

**deletecommand**(*name*)

Internal function.

Delete the Tcl command provided in NAME.
destroy()
    Destroy this and all descendants widgets.

event_add(virtual, *sequences)
    Bind a virtual event VIRTUAL (of the form <<Name>>) to an event SEQUENCE such that the virtual
    event is triggered whenever SEQUENCE occurs.

event_delete(virtual, *sequences)
    Unbind a virtual event VIRTUAL from SEQUENCE.

event_generate(sequence, **kw)
    Generate an event SEQUENCE. Additional keyword arguments specify parameter of the event (e.g. x, y,
    rootx, rooty).

event_info(virtual=None)
    Return a list of all virtual events or the information about the SEQUENCE bound to the virtual event
    VIRTUAL.

focus()
    Direct input focus to this widget.

    If the application currently does not have the focus this widget will get the focus if the application gets the
    focus through the window manager.

focus_displayof()
    Return the widget which has currently the focus on the display where this widget is located.

    Return None if the application does not have the focus.

focus_force()
    Direct input focus to this widget even if the application does not have the focus. Use with caution!

focus_get()
    Return the widget which has currently the focus in the application.

    Use focus_displayof to allow working with several displays. Return None if application does not have the
    focus.

focus_lastfor()
    Return the widget which would have the focus if top level for this widget gets the focus from the window
    manager.

focus_set()
    Direct input focus to this widget.

    If the application currently does not have the focus this widget will get the focus if the application gets the
    focus through the window manager.

focusmodel(model=None)
    Set focus model to MODEL. “active” means that this widget will claim the focus itself, “passive” means
    that the window manager shall give the focus. Return current focus model if MODEL is None.

frame()
    Return identifier for decorative frame of this widget if present.

geometry(newGeometry=None)
    Set geometry to NEWGEOMETRY of the form =widthxheight+x+y. Return current value if None is given.

getboolean(s)
    Return a boolean value for Tcl boolean values true and false given as parameter.

getdouble
    alias of float
getint
    alias of int

getvar (name='PY_VAR')
    Return value of Tcl variable NAME.

grab_current()
    Return widget which has currently the grab in this application or None.

grab_release()
    Release grab for this widget if currently set.

grab_set (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 descendants that don’t have an icon set explicitly. DEFAULT can be the relative path to a .ico file (example: root.iconbitmap(default=’myicon.ico’) ). See Tk documentation for more information.

iconify ()
Display widget as icon.

iconmask (bitmap=None)
Set mask for the icon bitmap of this widget. Return the mask if None is given.

iconname (newName=None)
Set the name of the icon for this widget. Return the name if None is given.

iconposition (x=None, y=None)
Set the position of the icon of this widget to X and Y. Return a tuple of the current values of X and X if None is given.

iconwindow (pathName=None)
Set widget PATHNAME to be displayed instead of icon. Return the current value if None is given.

image_names ()
Return a list of all existing image names.

image_types ()
Return a list of all available image types (e.g. photo bitmap).

keys ()
Return a list of all resource names of this widget.

lift (aboveThis=None)
Raise this widget in the stacking order.

lower (belowThis=None)
Lower this widget in the stacking order.

mainloop (n=0)
Call the mainloop of Tk.

maxsize (width=None, height=None)
Set max WIDTH and HEIGHT for this widget. If the window is gridded the values are given in grid units. Return the current values if None is given.

minsize (width=None, height=None)
Set min WIDTH and HEIGHT for this widget. If the window is gridded the values are given in grid units. Return the current values if None is given.

nametowidget (name)
Return the Tkinter instance of a widget identified by its Tcl name NAME.

option_add (pattern, value, priority=None)
Set a VALUE (second parameter) for an option PATTERN (first parameter).
An optional third parameter gives the numeric priority (defaults to 80).
option_clear()
Clear the option database.

It will be reloaded if option_add is called.

option_get(name, className)
Return the value for an option NAME for this widget with CLASSNAME.

Values with higher priority override lower values.

option_readfile(fileName, priority=None)
Read file FILENAME into the option database.

An optional second parameter gives the numeric priority.

overrideredirect(boolean=None)
Instruct the window manager to ignore this widget if BOOLEAN is given with 1. Return the current value
if None is given.

pack_propagate(flag=[])
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=[])
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 pixels. If there is no virtual root window return the width of the screen.

winfo_vrootx()
Return the x offset of the virtual root relative to the root window of the screen of this widget.

winfo_vrooty()
Return the y offset of the virtual root relative to the root window of the screen of this widget.

winfo_width()
Return the width of this widget.

winfo_x()
Return the x coordinate of the upper left corner of this widget in the parent.

winfo_y()
Return the y coordinate of the upper left corner of this widget in the parent.

withdraw()
Withdraw this widget from the screen such that it is unmapped and forgotten by the window manager. Re-draw it with wm_deiconify.

wm_aspect (minNumer=None, minDenom=None, maxNumer=None, maxDenom=None)
Instruct the window manager to set the aspect ratio (width/height) of this widget to be between MINNUMER/MINDENOM and MAXNUMER/MAXDENOM. Return a tuple of the actual values if no argument is given.

wm_attributes (*args)
This subcommand returns or sets platform specific attributes

The first form returns a list of the platform specific flags and their values. The second form returns the value for the specific option. The third form sets one or more of the values. The values are as follows:

On Windows, -disabled gets or sets whether the window is in a disabled state. -toolwindow gets or sets the style of the window to toolwindow (as defined in the MSDN). -topmost gets or sets whether this is a topmost window (displays above all other windows).

On Macintosh, XXXXX

On Unix, there are currently no special attribute values.

wm_client (name=None)
Store NAME in WM_CLIENT_MACHINE property of this widget. Return current value.

wm_colormapwindows (*wlist)
Store list of window names (WLIST) into WM_COLORMAPWINDOWS property of this widget. This list contains windows whose colormaps differ from their parents. Return current list of widgets if WLIST is empty.

wm_command (value=None)
Store VALUE in WM_COMMAND property. It is the command which shall be used to invoke the application. Return current command if VALUE is None.
wm_deiconify()
Deiconify this widget. If it was never mapped it will not be mapped. On Windows it will raise this widget and give it the focus.

wm_focusmodel(model=None)
Set focus model to MODEL. “active” means that this widget will claim the focus itself, “passive” means that the window manager shall give the focus. Return current focus model if MODEL is None.

wm_frame()
Return identifier for decorative frame of this widget if present.

wm_geometry(newGeometry=None)
Set geometry to NEWGEOMETRY of the form =widthxheight+x+y. Return current value if None is given.

wm_grid(baseWidth=None, baseHeight=None, widthInc=None, heightInc=None)
Instruct the window manager that this widget shall only be resized on grid boundaries. WIDTHINC and HEIGHTINC are the width and height of a grid unit in pixels. BASEWIDTH and BASEHEIGHT are the number of grid units requested in Tk_GeometryRequest.

wm_group(pathName=None)
Set the group leader widgets for related widgets to PATHNAME. Return the group leader of this widget if None is given.

wm_iconbitmap(bitmap=None, default=None)
Set bitmap for the iconified widget to BITMAP. Return the bitmap if None is given.
Under Windows, the DEFAULT parameter can be used to set the icon for the widget and any descendants that don’t have an icon set explicitly. DEFAULT can be the relative path to a .ico file (example: root.iconbitmap(default='myicon.ico')). See Tk documentation for more information.

wm_iconify()
Display widget as icon.

wm_iconmask(bitmap=None)
Set mask for the icon bitmap of this widget. Return the mask if None is given.

wm_iconname(newName=None)
Set the name of the icon for this widget. Return the name if None is given.

wm_iconposition(x=None, y=None)
Set the position of the icon of this widget to X and Y. Return a tuple of the current values of X and X if None is given.

wm_iconwindow(pathName=None)
Set widget PATHNAME to be displayed instead of icon. Return the current value if None is given.

wm_maxsize(width=None, height=None)
Set max WIDTH and HEIGHT for this widget. If the window is gridded the values are given in grid units. Return the current values if None is given.

wm_minsize(width=None, height=None)
Set min WIDTH and HEIGHT for this widget. If the window is gridded the values are given in grid units. Return the current values if None is given.

wm_overrideredirect(boolean=None)
Instruct the window manager to ignore this widget if BOOLEAN is given with 1. Return the current value if None is given.

wm_positionfrom(who=None)
Instruct the window manager that the position of this widget shall be defined by the user if WHO is “user”, and by its own policy if WHO is “program”. 

4.1. robot package
**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**.

```python
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 float
getint
alias of int

getvar(name='PY_VAR')
Return value of Tcl variable NAME.

grab_current()
Return widget which has currently the grab in this application or None.

grab_release()
Release grab for this widget if currently set.

grab_set(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 descendants that don’t have an icon set explicitly. DEFAULT can be the relative path to a .ico file (example: root.iconbitmap(default=’myicon.ico’)). See Tk documentation for more information.

iconify ()
Display widget as icon.

iconmask (bitmap=None)
Set mask for the icon bitmap of this widget. Return the mask if None is given.

iconname (newName=None)
Set the name of the icon for this widget. Return the name if None is given.

iconposition (x=None, y=None)
Set the position of the icon of this widget to X and Y. Return a tuple of the current values of X and X if None is given.

iconwindow (pathName=None)
Set widget PATHNAME to be displayed instead of icon. Return the current value if None is given.

image_names ()
Return a list of all existing image names.

image_types ()
Return a list of all available image types (e.g. photo bitmap).

keys ()
Return a list of all resource names of this widget.

lift (aboveThis=None)
Raise this widget in the stacking order.

lower (belowThis=None)
Lower this widget in the stacking order.

mainloop (n=0)
Call the mainloop of Tk.

maxsize (width=None, height=None)
Set max WIDTH and HEIGHT for this widget. If the window is gridded the values are given in grid units. Return the current values if None is given.

minsize (width=None, height=None)
Set min WIDTH and HEIGHT for this widget. If the window is gridded the values are given in grid units. Return the current values if None is given.

nametowidget (name)
Return the Tkinter instance of a widget identified by its Tcl name NAME.

option_add (pattern, value, priority=None)
Set a VALUE (second parameter) for an option PATTERN (first parameter).

An optional third parameter gives the numeric priority (defaults to 80).
option_clear()
Clear the option database.
It will be reloaded if option_add is called.

option_get(name, className)
Return the value for an option NAME for this widget with CLASSNAME.
Values with higher priority override lower values.

option_readfile(fileName, priority=None)
Read file FILENAME into the option database.
An optional second parameter gives the numeric priority.

overrideredirect(boolean=None)
Instruct the window manager to ignore this widget if BOOLEAN is given with 1. Return the current value if None is given.

pack_propagate(flag=["noarg"])
Set or get the status for propagation of geometry information.
A boolean argument specifies whether the geometry information of the slaves will determine the size of this widget. If no argument is given the current setting will be returned.

pack_slaves()
Return a list of all slaves of this widget in its packing order.

place_slaves()
Return a list of all slaves of this widget in its packing order.

positionfrom(who=None)
Instruct the window manager that the position of this widget shall be defined by the user if WHO is "user", and by its own policy if WHO is "program".

propagate(flag=["noarg"])
Set or get the status for propagation of geometry information.
A boolean argument specifies whether the geometry information of the slaves will determine the size of this widget. If no argument is given the current setting will be returned.

protocol(name=None, func=None)
Bind function FUNC to command NAME for this widget. Return the function bound to NAME if None is given. NAME could be e.g. "WM_SAVE_YOURSELF" or "WM_DELETE_WINDOW".

quit()
Quit the Tcl interpreter. All widgets will be destroyed.

register(func, subst=None, needcleanup=1)
Return a newly created Tcl function. If this function is called, the Python function FUNC will be executed. An optional function SUBST can be given which will be executed before FUNC.

resizable(width=None, height=None)
Instruct the window manager whether this width can be resized in WIDTH or HEIGHT. Both values are boolean values.

rowconfigure(index, cnf={}, **kw)
Configure row INDEX of a grid.
Valid resources are minsize (minimum size of the row), weight (how much does additional space propagate to this row) and pad (how much space to let additionally).

selection_clear(**kw)
Clear the current X selection.
**selection_get**( **kw** )
Return the contents of the current X selection.

A keyword parameter selection specifies the name of the selection and defaults to PRIMARY. A keyword parameter displayof specifies a widget on the display to use. A keyword parameter type specifies the form of data to be fetched, defaulting to STRING except on X11, where UTF8_STRING is tried before STRING.

**selection_handle**( command, **kw** )
Specify a function COMMAND to call if the X selection owned by this widget is queried by another application.

This function must return the contents of the selection. The function will be called with the arguments OFFSET and LENGTH which allows the chunking of very long selections. The following keyword parameters can be provided: selection - name of the selection (default PRIMARY), type - type of the selection (e.g. STRING, FILE_NAME).

**selection_own**( **kw** )
Become owner of X selection.

A keyword parameter selection specifies the name of the selection (default PRIMARY).

**selection_own_get**( **kw** )
Return owner of X selection.

The following keyword parameter can be provided: selection - name of the selection (default PRIMARY), type - type of the selection (e.g. STRING, FILE_NAME).

**send**( interp, cmd, *args )
Send Tcl command CMD to different interpreter INTERP to be executed.

**setvar**( name='PY_VAR', value='1' )
Set Tcl variable NAME to VALUE.

**show**( )

**size**( )
Return a tuple of the number of column and rows in the grid.

**sizefrom**( who=None )
Instruct the window manager that the size of this widget shall be defined by the user if WHO is “user”, and by its own policy if WHO is “program”.

**slaves**( )
Return a list of all slaves of this widget in its packing order.

**state**( newstate=None )
Query or set the state of this widget as one of normal, icon, iconic (see wm_iconwindow), withdrawn, or zoomed (Windows only).

**title**( string=None )
Set the title of this widget.

**tk_bisque**( )
Change the color scheme to light brown as used in Tk 3.6 and before.

**tk_focusFollowsMouse**( )
The widget under mouse will get automatically focus. Can not be disabled easily.

**tk_focusNext**( )
Return the next widget in the focus order which follows widget which has currently the focus.
The focus order first goes to the next child, then to the children of the child recursively and then to the next sibling which is higher in the stacking order. A widget is omitted if it has the takefocus resource set to 0.

\texttt{tk\_focusPrev()}  
Return previous widget in the focus order. See \texttt{tk\_focusNext} for details.

\texttt{tk\_menuBar(*args)}  
Do not use. Needed in Tk 3.6 and earlier.

\texttt{tk\_setPalette(*args, **kw)}  
Set a new color scheme for all widget elements.

A single color as argument will cause that all colors of Tk widget elements are derived from this. Alternatively several keyword parameters and its associated colors can be given. The following keywords are valid: activeBackground, foreground, selectColor, activeForeground, highlightBackground, selectForeground, background, highlightColor, selectForeground, disabledForeground, insertBackground, troughColor.

\texttt{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.

\texttt{tk\_raise(aboveThis=None)}  
Raise this widget in the stacking order.

\texttt{transient(master=None)}  
Instruct the window manager that this widget is transient with regard to widget MASTER.

\texttt{unbind(sequence, funcid=None)}  
Unbind for this widget for event SEQUENCE the function identified with FUNCID.

\texttt{unbind\_all(sequence)}  
Unbind for all widgets for event SEQUENCE all functions.

\texttt{unbind\_class(className, sequence)}  
Unbind for all widgets with bindtag CLASSNAME for event SEQUENCE all functions.

\texttt{update()}  
Enter event loop until all pending events have been processed by Tcl.

\texttt{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.

\texttt{wait\_variable(name='PY\_VAR')}  
Wait until the variable is modified.

A parameter of type IntVar, StringVar, DoubleVar or BooleanVar must be given.

\texttt{wait\_visibility(window=None)}  
Wait until the visibility of a WIDGET changes (e.g. it appears).

If no parameter is given self is used.

\texttt{wait\_window(window=None)}  
Wait until a WIDGET is destroyed.

If no parameter is given self is used.

\texttt{wait\_var(name='PY\_VAR')}  
Wait until the variable is modified.
A parameter of type IntVar, StringVar, DoubleVar or BooleanVar must be given.

\texttt{winfo\_atom(name, displayof=0)}
\begin{itemize}
  \item Return integer which represents atom NAME.
\end{itemize}

\texttt{winfo\_atomname(id, displayof=0)}
\begin{itemize}
  \item Return name of atom with identifier ID.
\end{itemize}

\texttt{winfo\_cells()}
\begin{itemize}
  \item Return number of cells in the colormap for this widget.
\end{itemize}

\texttt{winfo\_children()}
\begin{itemize}
  \item Return a list of all widgets which are children of this widget.
\end{itemize}

\texttt{winfo\_class()}
\begin{itemize}
  \item Return window class name of this widget.
\end{itemize}

\texttt{winfo\_colormapfull()}
\begin{itemize}
  \item Return true if at the last color request the colormap was full.
\end{itemize}

\texttt{winfo\_containing(rootX, rootY, displayof=0)}
\begin{itemize}
  \item Return the widget which is at the root coordinates ROOTX, ROOTY.
\end{itemize}

\texttt{winfo\_depth()}
\begin{itemize}
  \item Return the number of bits per pixel.
\end{itemize}

\texttt{winfo\_exists()}
\begin{itemize}
  \item Return true if this widget exists.
\end{itemize}

\texttt{winfo\_fpixels(number)}
\begin{itemize}
  \item Return the number of pixels for the given distance NUMBER (e.g. “3c”) as float.
\end{itemize}

\texttt{winfo\_geometry()}
\begin{itemize}
  \item Return geometry string for this widget in the form “widthxheight+X+Y”.
\end{itemize}

\texttt{winfo\_height()}
\begin{itemize}
  \item Return height of this widget.
\end{itemize}

\texttt{winfo\_id()}
\begin{itemize}
  \item Return identifier ID for this widget.
\end{itemize}

\texttt{winfo\_interps(displayof=0)}
\begin{itemize}
  \item Return the name of all Tcl interpreters for this display.
\end{itemize}

\texttt{winfo\_ismapped()}
\begin{itemize}
  \item Return true if this widget is mapped.
\end{itemize}

\texttt{winfo\_manager()}
\begin{itemize}
  \item Return the window manager name for this widget.
\end{itemize}

\texttt{winfo\_name()}
\begin{itemize}
  \item Return the name of this widget.
\end{itemize}

\texttt{winfo\_parent()}
\begin{itemize}
  \item Return the name of the parent of this widget.
\end{itemize}

\texttt{winfo\_pathname(id, displayof=0)}
\begin{itemize}
  \item Return the pathname of the widget given by ID.
\end{itemize}

\texttt{winfo\_pixels(number)}
\begin{itemize}
  \item Rounded integer value of winfo\_fpixels.
\end{itemize}

\texttt{winfo\_pointerx()}
\begin{itemize}
  \item Return the x coordinate of the pointer on the root window.
\end{itemize}
**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 appli-
   cation. Return current command if VALUE is None.
**wm_deiconify()**

   Deiconify this widget. If it was never mapped it will not be mapped. On Windows it will raise this widget and give it the focus.

**wm_focusmodel (model=None)**

   Set focus model to MODEL. “active” means that this widget will claim the focus itself, “passive” means that the window manager shall give the focus. Return current focus model if MODEL is None.

**wm_frame ()**

   Return identifier for decorative frame of this widget if present.

**wm_geometry (newGeometry=None)**

   Set geometry to NEWGEOMETRY of the form =widthxheight+x+y. Return current value if None is given.

**wm_grid (baseWidth=None, baseHeight=None, widthInc=None, heightInc=None)**

   Instruct the window manager that this widget shall only be resized on grid boundaries. WIDTHINC and HEIGHTINC are the width and height of a grid unit in pixels. BASEWIDTH and BASEHEIGHT are the number of grid units requested in Tk_GeometryRequest.

**wm_group (pathName=None)**

   Set the group leader widgets for related widgets to PATHNAME. Return the group leader of this widget if None is given.

**wm_iconbitmap (bitmap=None, default=None)**

   Set bitmap for the iconified widget to BITMAP. Return the bitmap if None is given.

   Under Windows, the DEFAULT parameter can be used to set the icon for the widget and any descendants that don’t have an icon set explicitly. DEFAULT can be the relative path to a .ico file (example: root.iconbitmap(default=’myicon.ico’)). See Tk documentation for more information.

**wm_iconify ()**

   Display widget as icon.

**wm_iconmask (bitmap=None)**

   Set mask for the icon bitmap of this widget. Return the mask if None is given.

**wm_iconname (newName=None)**

   Set the name of the icon for this widget. Return the name if None is given.

**wm_iconposition (x=None, y=None)**

   Set the position of the icon of this widget to X and Y. Return a tuple of the current values of X and X if None is given.

**wm_iconwindow (pathName=None)**

   Set widget PATHNAME to be displayed instead of icon. Return the current value if None is given.

**wm_maxsize (width=None, height=None)**

   Set max WIDTH and HEIGHT for this widget. If the window is gridded the values are given in grid units. Return the current values if None is given.

**wm_minsize (width=None, height=None)**

   Set min WIDTH and HEIGHT for this widget. If the window is gridded the values are given in grid units. Return the current values if None is given.

**wm_overrideredirect (boolean=None)**

   Instruct the window manager to ignore this widget if BOOLEAN is given with 1. Return the current value if None is given.

**wm_positionfrom (who=None)**

   Instruct the window manager that the position of this widget shall be defined by the user if WHO is “user”, and by its own policy if WHO is “program”.

---

154 Chapter 4. All packages
wm_protocol (name=None, func=None)

Bind function FUNC to command NAME for this widget. Return the function bound to NAME if None is given. NAME could be e.g. “WM_SAVE_YOURSELF” or “WM_DELETE_WINDOW”.

wm_resizable (width=None, height=None)

Instruct the window manager whether this width can be resized in WIDTH or HEIGHT. Both values are boolean values.

wm_sizefrom (who=None)

Instruct the window manager that the size of this widget shall be defined by the user if WHO is “user”, and by its own policy if WHO is “program”.

wm_state (newstate=None)

Query or set the state of this widget as one of normal, icon, iconic (see wm_iconwindow), withdrawn, or zoomed (Windows only).

wm_title (string=None)

Set the title of this widget.

wm_transient (master=None)

Instruct the window manager that this widget is transient with regard to widget MASTER.

wm_withdraw()

Withdraw this widget from the screen such that it is unmapped and forgotten by the window manager. Re-draw it with wm_deiconify.

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)

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)
visit_test (test)

visit_keyword (kw)

debug_keyword (keyword)
    Called when keyword ends. Default implementation does nothing.

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

debug_test (test)
    Called when test ends. Default implementation does nothing.

canary_keyword (keyword)
    Called when keyword starts. Default implementation does nothing.
    Can return explicit False to stop visiting.

canary_message (msg)
    Called when message starts. Default implementation does nothing.
    Can return explicit False to stop visiting.

canary_suite (suite)
    Called when suite starts. Default implementation does nothing.
    Can return explicit False to stop visiting.

canary_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)

    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)
end_test \(\text{(test)}\)
   Called when test ends. Default implementation does nothing.

start_keyword \(\text{(keyword)}\)
   Called when keyword starts. Default implementation does nothing.
   Can return explicit False to stop visiting.

start_message \(\text{(msg)}\)
   Called when message starts. Default implementation does nothing.
   Can return explicit False to stop visiting.

start_test \(\text{(test)}\)
   Called when test starts. Default implementation does nothing.
   Can return explicit False to stop visiting.

visit_keyword \(\text{(kw)}\)
visit_message \(\text{(msg)}\)
   Implements visiting the message.
   Can be overridden to allow modifying the passed in msg without calling start_message() or end_message().

visit_suite \(\text{(suite)}\)
   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 \(\text{(test)}\)

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 \(\text{(message, level='ERROR')}\)

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.
   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
      Internal usage only.
      alias of Message
   doc
   args
      Keyword arguments as a list of strings.

4.1. robot package
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.

deepecopy (**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 ()
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()` \(\rightarrow (k, v)\), remove and return some (key, value) pair as a 2-tuple; but raise KeyError if D is empty.

`setdefault` \([(k, d)]\) \(\rightarrow D\.get(k, d)\), also set \(D[k]=d\) if \(k\) not in \(D\)

`update` \([E], **F\) \(\rightarrow None\). 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()` \(\rightarrow\) 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
    Bases: robot.model.visitor.SuiteVisitor

    `visit_suite` (suite)

    `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.

4.1. robot package
**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)`
**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)
    end_suite (suite)
    visit_test (test)
    visit_keyword (kw)
    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. sl-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)
visit_test(test)
visit_keyword(keyword)
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=None, doc=None, 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
    Internal usage only

    alias of 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.

deepe copy (**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)
class robot.model.testsuite.TestSuite(name=", doc=", metadata=None, source=None)

Bases: robot.model.modelobject.ModelObject

Base model for single suite.

Extended by robot.running.model.TestSuite and robot.result.model.TestSuite.

test_class
    Internal usage only.
    alias of TestCase

keyword_class
    Internal usage only.
    alias of Keyword

parent
    Parent suite. None with the root suite.

doc
    Test suite documentation.

source
    Path to the source file or directory.

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.

Example:

```python
suite.filter(included_tests=['Test 1', '* Example'], included_tags='priority-1')
```

**configure** (**options**)

A shortcut to configure a suite using one method call.

**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

```python
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`

Bases: `object`

Container for total statistics.

**critical** = `None`

Instance of `TotalStat` for critical tests.

**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)*

Bases: `robot.model.visitor.SuiteVisitor`

**add_test** *(test)*

**visit_test** *(test)*

**visit_keyword** *(kw)*

**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`.

```python
class robot.model.visitor.SuiteVisitor
    Bases: object
```

4.1. robot package
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)`
- `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
    by_method_name(name, arguments)
    get_arguments(version)

class robot.output.listenerarguments.StartSuiteArguments(arguments)
    Bases: robot.output.listenerarguments._ListenerArgumentsFromItem
    by_method_name(name, arguments)
    get_arguments(version)

class robot.output.listenerarguments.EndSuiteArguments(arguments)
    Bases: robot.output.listenerarguments.StartSuiteArguments
    by_method_name(name, arguments)
    get_arguments(version)

class robot.output.listenerarguments.StartTestArguments(arguments)
    Bases: robot.output.listenerarguments._ListenerArgumentsFromItem
    by_method_name(name, arguments)
    get_arguments(version)

class robot.output.listenerarguments.EndTestArguments(arguments)
    Bases: robot.output.listenerarguments.StartTestArguments
    by_method_name(name, arguments)
    get_arguments(version)

class robot.output.listenerarguments.StartKeywordArguments(arguments)
    Bases: robot.output.listenerarguments._ListenerArgumentsFromItem
    by_method_name(name, arguments)
    get_arguments(version)

class robot.output.listenerarguments.EndKeywordArguments(arguments)
    Bases: robot.output.listenerarguments.StartKeywordArguments
    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', std-
out=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, ...

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
timestamp
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.

```python
html
html_message
   Returns the message content as HTML.
level
parent
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=..., prefix=...)
   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)
```

4.1. robot package
robot.output.pyloggingconf module

```python
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)
    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
```

Chapter 4. All packages
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.

setName(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', generator='Robot')
    Bases: robot.result.visitor.ResultVisitor
    close()
    set_log_level(level)
    message(msg)
    log_message(msg)
    start_keyword(kw)
    end_keyword(kw)
    start_test(test)
    end_test(test)
    start_suite(suite)
    end_suite(suite)
    start_statistics(stats)
    end_statistics(stats)
    start_total_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**

```python
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)
```

The `robot.parsing.disable_curdir_processing` method

Decorator to disable processing `${CURDIR}` variable.

### Submodules

**robot.parsing.comments module**

```python
class robot.parsing.comments.CommentCache
    Bases: object
    add(comment)
    consume_with(function)

class robot.parsing.comments.Comments
    Bases: object
    add(row)
    value

class robot.parsing.comments.Comment(comment_data)
    Bases: object
    as_list()
```

**robot.parsing.datarow module**

```python
class robot.parsing.datarow.DataRow(cells)
    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)
    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=False, 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 – Boolean to control warning about skipped files.

• extensions – List/set of extensions to parse. If None, all files supported by Robot Frame-
   work 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 hierachical 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.

```python
populate(include_suites=None, warn_on_skipped=False, extensions=None, recurse=True)
```

```python
add_child(path, include_suites, extensions=None, warn_on_skipped=False)
```

```python
has_tests()
```

```python
imports
```

```python
keywords
```

```python
name
```

```python
report_invalid_syntax(message, level='ERROR')
```

```python
save(**options)
```

This writes the datafile to disk.

**Parameters options** – Configuration for writing. These are passed to `WritingContext` as keyword arguments.

See also `robot.writer.datafilewriter.DataFileWriter`

```python
start_table(header_row)
```

**class** `robot.parsing.model_TestCaseFileSettingTable` *(parent)*

Bases: `robot.parsing.model._SettingTable`

```python
add_library(name, args=None, comment=None)
```

```python
add_metadata(name, value='', comment=None)
```

```python
add_resource(name, invalid_args=None, comment=None)
```

```python
add_variables(name, args=None, comment=None)
```

```python
directory
```

```python
get_setter(setting_name)
```

```python
header
```

```python
is_setting(setting_name)
```

```python
name
```

```python
normalize(setting)
```

```python
report_invalid_syntax(message, level='ERROR')
```

```python
set_header(header)
```

```python
source
```

```python
type = 'setting'
```

**class** `robot.parsing.model_ResourceFileSettingTable` *(parent)*

Bases: `robot.parsing.model._SettingTable`

```python
add_library(name, args=None, comment=None)
```

```python
add_metadata(name, value='', comment=None)
```

```python
add_resource(name, invalid_args=None, comment=None)
```

```python
add_variables(name, args=None, comment=None)
```

```python
directory
```

4.1. robot package
get_setter(setting_name)
header
is_setting(setting_name)
name
normalize(setting)
report_invalid_syntax(message, level='ERROR')
set_header(header)
source
type = 'setting'
class robot.parsing.model.InitFileSettingTable(parent)
Bases: robot.parsing.model._SettingTable
add_library(name, args=None, comment=None)
add_metadata(name, value='', comment=None)
add_resource(name, invalid_args=None, comment=None)
add_variables(name, args=None, comment=None)
directory
get_setter(setting_name)
header
is_setting(setting_name)
name
normalize(setting)
report_invalid_syntax(message, level='ERROR')
set_header(header)
source
type = 'setting'
class robot.parsing.model.VariableTable(parent)
Bases: robot.parsing.model._Table
type = 'variable'
add(name, value, comment=None)
directory
header
name
report_invalid_syntax(message, level='ERROR')
set_header(header)
source
class robot.parsing.model.TestCaseTable(parent)
Bases: robot.parsing.model._Table
type = 'test case'
add(name)
is_started()
directory
header
name
report_invalid_syntax(message, level='ERROR')
set_header(header)
source
class robot.parsing.model.KeywordTable (parent)
   Bases: robot.parsing.model._Table
   type = 'keyword'
   add(name)
directory
header
name
report_invalid_syntax(message, level='ERROR')
set_header(header)
source
class robot.parsing.model.Variable (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)
report_invalid_syntax(message, level='ERROR')
settings
add_step(content, comment=None)
copy(name)
get_setter(setting_name)
is_setting(setting_name)
normalize(setting)

class robot.parsing.model.UserKeyword(parent, name)
    Bases: robot.parsing.model.TestCase
    settings
    add_for_loop(declaration, comment=None)
    add_step(content, comment=None)
    copy(name)
    directory
    get_setter(setting_name)
    is_setting(setting_name)
    normalize(setting)
    report_invalid_syntax(message, level='ERROR')
    source

class robot.parsing.model.ForLoop(declaration, comment=None)
    Bases: robot.parsing.model._WithSteps
    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.

    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)

```python
class robot.parsing.model.OldStyleSettingAndVariableTableHeaderMatcher
    Bases: object
    match (header)

class robot.parsing.model.OldStyleTestAndKeywordTableHeaderMatcher
    Bases: object
    match (header)
```

**robot.parsing.populators module**

```python
class robot.parsing.populators.FromFilePopulator (datafile)
    Bases: object
    populate (path)
    start_table (header)
    eof ()
    add (row)

class robot.parsing.populators.FromDirectoryPopulator
    Bases: object
    ignored_prefixes = ('_', '．')
    ignored_dirs = ('CVS',)
    populate (path, datadir, include_suites=None, warn_on_skipped=False, include_extensions=None, recurse=True)
```

**robot.parsing.restreader module**

```python
robot.parsing.restreader.RestReader()
```

**robot.parsing.restsupport module**

```python
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.

```python
definition
final_argument_whitespace = False
has_content = True
info(message)
option_spec = {'number-lines': <function unchanged>, 'class': <function class_option>}
optional_arguments = 1
required_arguments = 0
severe(message)
warning(message)
```

**class** robot.parsing.restsupport.RobotDataStorage(doctree)

```python
Bases: object
add_data(rows)
get_data()
has_data()
```

**robot.parsing.settings module**

```python
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)

```python
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_set()

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.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.
```python
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()
```

202 Chapter 4. All packages
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: object
    read(tsvfile, populator)
    classmethod split_row(row)

robot.parsing.txtreader module

class robot.parsing.txtreader.TxtReader
    Bases: robot.parsing.tsvreader.TsvReader
    classmethod split_row(row)
    read(tsvfile, populator)

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 (*args, **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)
    Bases: robot.output.xmllogger.XmlLogger
    start_message(msg)
    visit_keyword(kw)
    close()
end_result (result)
end_errors (errors=None)
end_keyword (kw)
end_message (msg)
   Called when message ends. Default implementation does nothing.
end_stat (stat)
end_statistics (stats)
end_suite (suite)
end_suite_statistics (tag_stats)
end_tag_statistics (tag_stats)
end_test (test)
end_total_statistics (total_stats)
log_message (msg)
message (msg)
set_log_level (level)
start_errors (errors=None)
start_keyword (kw)
start_result (result)
start_stat (stat)
start_statistics (stats)
start_suite (suite)
start_suite_statistics (tag_stats)
start_tag_statistics (tag_stats)
start_test (test)
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)
end_suite(suite)
visit_test(test)
visit_keyword(kw)
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.
Robot Framework Documentation, Release 3.1.0.dev1

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:])
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

4.1. robot package

211


visit_suite (suite)

add_tags

definitions

called when definitions ends. Default implementation does nothing.

definitions

called when definitions starts. Default implementation does nothing.

definitions

called when definitions 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.

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 Message
messages
A list-like object of Message instances.
add(other)
visit(visitor)
```

**robot.result.executionresult module**

class robot.result.executionresult.Result

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**

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.

---

**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.

`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)
    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)

visit_test(test)

visit_keyword(keyword)

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)
    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)

   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.

```python
class robot.result.keywordremover.ForLoopItemsRemover
    Bases: robot.result.keywordremover._KeywordRemover

    start_keyword(kw)
    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.
```

```
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)

start_test(test)

start_keyword(keyword)

visit_message(msg)

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)

start_test(test)

start_keyword(keyword)

visit_message(msg)

end_keyword(keyword)
Called when keyword ends. Default implementation does nothing.
Robot Framework Documentation, Release 3.1.0.dev1

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

default_suite (suite)
Called when suite ends. Default implementation does nothing.

default_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)

default_suite (suite)

end_suite (suite)

visit_test (test)

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)*

**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

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.
depcopy (**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.

deprecated(**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.

```python
class robot.result.model.TestSuite(name=", doc="", metadata=None, source=None, message="", starttime=None, endtime=None)
```

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**
'SPASS' 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:

```python
stats = suite.statistics
print stats.critical.failed
print stats.all.total
print stats.message
```

**full_message**
Combination of `message` and `stat_message`.

---

4.1. robot package 227
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 documenta-
tion 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.

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 ex-
ample, 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

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:

```python
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.

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. 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)
Robot Framework Documentation, Release 3.1.0.dev1

```python
visit_test(test)
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.
```
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)

visit_keyword (keyword)

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.

```python
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)

dend(elem)

class robot.result.xmlelementhandlers.RootHandler
Bases: robot.result.xmlelementhandlers._Handler

dend(elem, result)

get_child_handler(elem)

start(elem, result)

class robot.result.xmlelementhandlers.RobotHandler
Bases: robot.result.xmlelementhandlers._Handler

tag = 'robot'

start(elem, result)

dend(elem, result)

get_child_handler(elem)

class robot.result.xmlelementhandlers.SuiteHandler
Bases: robot.result.xmlelementhandlers._Handler

tag = 'suite'

start(elem, result)

dend(elem, result)

get_child_handler(elem)

class robot.result.xmlelementhandlers.RootSuiteHandler
Bases: robot.result.xmlelementhandlers.SuiteHandler

start(elem, result)

dend(elem, result)

get_child_handler(elem)

tag = 'suite'

class robot.result.xmlelementhandlers.TestCaseHandler
Bases: robot.result.xmlelementhandlers._Handler

tag = 'test'

4.1. robot package
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)
```python
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)
```

4.1. robot package
class robot.result.xml_elementhandlers.AssignVarHandler
    Bases: robot.result.xml_elementhandlers._Handler
    
    tag = 'var'
    
    def __init__(self, elem, result):
        get_child_handler(elem)

    def start(self, elem, result):

class robot.result.xml_elementhandlers.ArgumentsHandler
    Bases: robot.result.xml_elementhandlers._Handler
    
    tag = 'arguments'
    
    def __init__(self, elem, result):
        get_child_handler(elem)

    def start(self, elem, result):

class robot.result.xml_elementhandlers.ArgumentHandler
    Bases: robot.result.xml_elementhandlers._Handler
    
    tag = 'arg'
    
    def __init__(self, elem, result):
        get_child_handler(elem)

    def start(self, elem, result):

class robot.result.xml_elementhandlers.ErrorsHandler
    Bases: robot.result.xml_elementhandlers._Handler
    
    tag = 'errors'
    
    def __init__(self, elem, result):
        get_child_handler(elem)

    def start(self, elem, result):

class robot.result.xml_elementhandlers.StatisticsHandler
    Bases: robot.result.xml_elementhandlers._Handler
    
    tag = 'statistics'
    
    def __init__(self, elem, result):
        get_child_handler(elem)

    def start(self, 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:

- :class:`TestSuiteBuilder` for creating executable test suites based on existing test case files and directories.
- :class:`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.

This package and especially all public code was rewritten in Robot Framework 2.8 to make it easier to generate and execute test suites programmatically. Rewriting of the test execution logic will continue in future releases, but changes to the public API ought to be relatively small.

Examples

First, let's assume we have the following test suite in file activate_skynet.robot:

```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:

```python
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:

```python
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:

```python
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

class robot.running.arguments.argumentparser.PythonArgumentParser(type='Keyword')
    Bases: robot.running.arguments.argumentparser._ArgumentParser
    parse(source, name=None)

class robot.running.arguments.argumentparser.JavaArgumentParser(type='Keyword')
    Bases: robot.running.arguments.argumentparser._ArgumentParser
    parse(source, 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 Framework Documentation, Release 3.1.0.dev1

robot.running.arguments.argumentresolver module

```python
class robot.running.arguments.argumentresolver.ArgumentResolver (argspec,  
resolve_named=True,  
resolve_variables_until=None,  
dict_to_kwargs=False)

Bases: object

resolve (arguments, variables=None)
```

```python
class robot.running.arguments.argumentresolver.NamedArgumentResolver (argspec)
Bases: object

resolve (arguments, variables=None)
```

```python
class robot.running.arguments.argumentresolver.NullNamedArgumentResolver
Bases: object

resolve (arguments, variables=None)
```

```python
class robot.running.arguments.argumentresolver.DictToKwargs (argspec,  
enabled=False)
Bases: object

handle (positional, named)
```

```python
class robot.running.arguments.argumentresolver.VariableReplacer (resolve_until=None)
Bases: object

replace (positional, named, variables=None)
```

robot.running.arguments.argumentspec module

```python
class robot.running.arguments.argumentspec.ArgumentSpec (name=None,  
type='Keyword',  
positionals=None,  
defaults=None,  
varargs=None,  
kwargs=None,  
supports_named=True)

Bases: object

minargs

maxargs

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

```python
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

class robot.running.arguments.javaargumentcoercer.JavaArgumentCoercer(signatures, argspec)
    Bases: object
    coerce(arguments, named, dryrun=False)

class robot.running.arguments.javaargumentcoercer.CoercerFinder
    Bases: object
    find_coercers(signatures)

class robot.running.arguments.javaargumentcoercer.BooleanCoercer(position=None)
    Bases: robot.running.arguments.javaargumentcoercer._Coercer
    coerce(argument, dryrun=False)
    handles(type)

class robot.running.arguments.javaargumentcoercer.IntegerCoercer(position=None)
    Bases: robot.running.arguments.javaargumentcoercer._Coercer
    coerce(argument, dryrun=False)
    handles(type)

class robot.running.arguments.javaargumentcoercer.FloatCoercer(position=None)
    Bases: robot.running.arguments.javaargumentcoercer._Coercer
    coerce(argument, dryrun=False)
    handles(type)

class robot.running.arguments.javaargumentcoercer.NullCoercer(position=None)
    Bases: robot.running.arguments.javaargumentcoercer._Coercer
    handles(argument)
    coerce(argument, dryrun=False)

class robot.running.arguments.javaargumentcoercer.VarargsHandler(argspec)
    Bases: object
    handle(Arguments)

robot.running.timeouts package

class robot.running.timeouts.TestTimeout(timeout=None, message=", variables=None)
    Bases: robot.running.timeouts._Timeout
    type = 'Test'
set_keyword_timeout (timeout_occurred)

any_timeout_occurred()

active

get_message()

replace_variables (variables)

run (runnable, args=None, kwargs=None)

start ()

time_left ()

timed_out ()

class robot.running.timeouts.KeywordTimeout (timeout=None, message=", variables=None)

Bases: robot.running.timeouts._Timeout

type = 'Keyword'

active

get_message()

replace_variables (variables)

run (runnable, args=None, kwargs=None)

start ()

time_left ()

timed_out ()

Submodules

robot.running.timeouts.ironpython module

class robot.running.timeouts.ironpython.Timeout (timeout, error)

Bases: object

execute (runnable)

class robot.running.timeouts.ironpython.Runner (runnable)

Bases: object

get_result ()

robot.running.timeouts.jython module

class robot.running.timeouts.jython.Timeout (timeout, error)

Bases: object

execute (runnable)
robot.running.timeouts.posix module

class robot.running.timeouts.posix.Timeout (timeout, error)
    Bases: object
    execute (Runnable)

gerobot.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=False, extension=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 – Boolean to control should a warning be emitted if a file is skipped
          because it cannot be parsed or should it be ignored silently. When executing tests normally,
          this value is set with the --warnonskippedfiles option.
        • 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.

    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')
built_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.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
    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)
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 ()

4.1. robot package
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.

```python
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.

```python
keyword_class = None
```

Child keywords as a `Keywords` object.

```python
messages
```

Messages as a `Messages` object.

```python
name
```

```python
parent
```

Parent test suite, test case or keyword.

```python
tags
```

Keyword tags as a `Tags` object.

```python
timeout
```

type

```python
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`.

```python
keyword_class
```

Internal usage only.

```python
flavor
```

```python
variables
```

```python
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.

deepecopy (**attributes)

Return deep copy of this object.

Parameters attributes – Attributes to be set for the returned copy automatically. For example, test.deepecopy(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.

deprecatedcopy(**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)
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.

imports
Deprecated. Use TestSuite.resource.imports instead.

variables
Deprecated. Use TestSuite.resource.variables instead.

user_keywords
Deprecated. Use TestSuite.resource.keywords instead.

configure(randomize_suites=False, randomize_tests=False, randomize_seed=None, **options)
A shortcut to configure a suite using one method call.

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 on 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, and they also have the same semantics. 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. Note that this works only in version 2.8.4 and newer.

Only options related to the actual test execution have an effect. For example, options related to selecting test cases 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:**

```python
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.

```python
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.

**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()

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)
    visit_test(test)
    visit_keyword(kw)
    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)

**end_suite**(suite)

**visit_test**(test)

**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 \texttt{kw} without calling \texttt{start_keyword()} or \texttt{end_keyword()} nor visiting child keywords.

\texttt{visit\_message}(msg)
Implements visiting the message.
Can be overridden to allow modifying the passed in \texttt{msg} without calling \texttt{start_message()} or \texttt{end_message()}.

\texttt{visit\_suite}(suite)
Implements traversing through the suite and its direct children.
Can be overridden to allow modifying the passed in \texttt{suite} without calling \texttt{start_suite()} or \texttt{end_suite()} nor visiting child suites, tests or keywords (setup and teardown) at all.

\begin{verbatim}
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()

\end{verbatim}
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:
%s'
teardown_message = 'Teardown failed:
%s'
also_teardown_message = '%s
Also teardown failed:
%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 use.'
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:
%s'
teardown_message = 'Suite teardown failed:
%s'
also_teardown_message = '%s
Also suite teardown failed:
%s'
message

class robot.running.status/ParentMessage(status)
Bases: robot.running.status/SuiteMessage
setup_message = 'Parent suite setup failed:
%s'
teardown_message = 'Parent suite teardown failed:
%s'
also_teardown_message = '%s
Also parent suite teardown failed:
%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.

```python
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 StatusRe-
    porter(...)” 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
    longname
    shortdoc
    matches (name)
    create_runner (name)

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 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 AssertionsErrors occur outside.

Most of the functions are copied more or less directly from unittest.TestCase which comes with the following license. Further information about unittest in general can be found from http://pyunit.sourceforge.net/. This module can be used freely in same terms as unittest.

unittest license:

Copyright (c) 1999-2003 Steve Purcell
This module is free software, and you may redistribute it and/or modify it under the same terms as Python itself, so long as this copyright message and disclaimer are retained in their original form.

IN NO EVENT SHALL THE AUTHOR BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF THIS CODE, EVEN IF THE AUTHOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

THE AUTHOR SPECIFICALLY DISCLAIMS ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE CODE PROVIDED HEREUNDER IS ON AN "AS IS" BASIS, AND THERE IS NO OBLIGATION WHATSOEVER TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

Examples:

```python
import unittest
from robot.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
======================================================================
```

264 Chapter 4. All packages
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

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)
    Fail if given objects are unequal as determined by the ‘==’ operator.

robot.utils.asserts.assert_not_equal (first, second, msg=None, values=True)
    Fail if given objects are equal as determined by the ‘!=' operator.

robot.utils.asserts.assert_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.


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.')</class>

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.

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

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 Iron-  
    Python 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.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()
    format_line(line)
    add(line)
    end()
    format(lines)
    handles(line)
class robot.utils.htmlformatters.HeaderFormatter
    Bases: robot.utils.htmlformatters._SingleLineFormatter
    match()
    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)
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.

class robot.utils.markupwriters.NullMarkupWriter(*args, **kwargs)
Bases: object

Null implementation of the _MarkupWriter interface.

start(*args, **kwargs)
content(*args, **kwargs)
element(*args, **kwargs)
end(*args, **kwargs)
close(*args, **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.


If ‘code_style’ is True:

‘mixedCAPSCamel’ -> ‘Mixed CAPS Camel’ ‘camelCaseName’ -> ‘Camel Case Name’ ‘under_score_name’ -> ‘Under Score Name’ ‘under_and space’ -> ‘Under And Space’ ‘miXed CAPS_nAME’ -> ‘MiXed CAPS NAME’ ‘ ’ -> ‘ ’

```
robot.utils.misc.plural_or_not(item)
```

```
robot.utils.misc.seq2str(sequence, quote=''', sep=', ', lastsep=' and ')
```

Returns sequence in format ‘item 1’, ‘item 2’ and ‘item 3’.

```
robot.utils.misc.seq2str2(sequence)
```

Returns sequence in format [ item 1 | item 2 | . . . ].

```
robot.utils.misc.getdoc(item)
```

```
robot.utils.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()
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
```
**Class: robot.utils.ordereddict.OrderedDict**

```python
class robot.utils.ordereddict.OrderedDict(*args, **kwds)
    Bases: dict, UserDict.DictMixin

    clear()
    popitem(last=True)
    keys()
    setdefault(key, default=None)
    update(other=None, **kwargs)
    pop(key, *args)
    values()
    items()
    iterkeys()
    itervalues()
    iteritems()
    copy()
    classmethod fromkeys(iterable, value=None)
    get(key, 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
    viewitems() → a set-like object providing a view on D’s items
    viewkeys() → a set-like object providing a view on D’s keys
    viewvalues() → an object providing a view on D’s values
```

**Class: robot.utils.recommendations.RecommendationFinder**

```python
class robot.utils.recommendations.RecommendationFinder(normalizer=None)
    Bases: object
```

---

4.1. robot package
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.


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.
4. Handle non-ASCII characters on working directory with Python < 2.6.5: http://bugs.python.org/issue3426
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 maximum of 23 and minutes and seconds both have 59 (e.g. ‘1 minute 40 seconds’ instead
  of ‘100 seconds’)

If compact has value ‘True’, short suffixes are used. (e.g. 1d 2h 3min 4s 5ms)

robot.utils.robottime.format_time(timetuple_or_epochsecs, daysep=",", daytimesep=’,’, timeseq=’’;
  millissep=None, gmtsep=None)

Returns a timestamp formatted from given time using separators.

Time can be given either as a timetuple or seconds after epoch.

Timetuple is (year, month, day, hour, min, sec[, millis]), where parts must be integers and millis is required
only when millissep is not None. Notice that this is not 100% compatible with standard Python timetuples which
do not have millis.

Seconds after epoch can be either an integer or a float.

robot.utils.robottime.get_time(format='timestamp', time_=None)

Return the given or current time in requested format.

If time is not given, current time is used. How time is returned is 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’).

4.1. robot package
3. ‘NOW’ (case-insensitive) is the current local time.

4. ‘UTC’ (case-insensitive) is the current time in UTC.

5. Format ‘NOW - 1 day’ or ‘UTC + 1 hour 30 min’ is the current local/UTC time plus/minus the time specified with the time string.

Seconds are rounded down to avoid getting times in the future.

```python
robot.utils.robottime.get_timestamp(daysep=",", daytimesep=" ", timesep=":", millissep=".")
```

```python
robot.utils.robottime.timestamp_to_secs(timestamp, seps=None)
```

```python
robot.utils.robottime.secs_to_timestamp(secs, seps=None, millis=False)
```

```python
robot.utils.robottime.get_elapsed_time(start_time, end_time)
```

Returns the time between given timestamps in milliseconds.

```python
robot.utils.robottime.elapsed_time_to_string(elapsed, include_millis=True)
```

If `include_millis` is True, ‘.mil’ part is omitted.

```python
class robot.utils.robottime.TimestampCache
    Bases: object
    
    get_timestamp(daysep=",", daytimesep=" ", timesep=":", millissep=".")
```

**robot.utils.robottypes module**

```python
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 `True` if it is not `FALSE`, `NO`, ‘NONE’ or ‘’, case-insensitively. Considering ‘NONE’ not `True` is new in RF 3.0.3.
- 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()` and `is_noney()`.

```python
robot.utils.robottypes.is_falsy(item)
```

Opposite of `is_truthy()`.

**robot.utils.robottypes2 module**

```python
robot.utils.robottypes2.is_integer(item)
```

```python
robot.utils.robottypes2.is_number(item)
```

```python
robot.utils.robottypes2.is_bytes(item)
```

```python
robot.utils.robottypes2.is_string(item)
```

```python
robot.utils.robottypes2.is_unicode(item)
```

```python
robot.utils.robottypes2.is_list_like(item)
```

```python
robot.utils.robottypes2.is_dict_like(item)
```

```python
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.unic module

robot.utils.unic.unic (item)
robot.utils.unic.prepr (item, width=400)
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 = '$@&'
    find (key)

class robot.variables.finders.ExtendedFinder (finder)
    Bases: object
    identifiers = '$@&'
    find (name)

class robot.variables.finders.EnvironmentFinder
    Bases: object
    identifiers = '%$'

4.1. robot package
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()
end_keyword()
set_global(name, value)
set_suite(name, value)
Robot Framework Documentation, Release 3.1.0.dev1

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*:

```python
WriteConfiguration(datafile, output=StringIO)
```

Output file is created from *datafile.source* by stripping extension and replacing it with *html*:

```python
WriteConfiguration(datafile, format='html')
```

```python
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.

```python
class robot.writer.filewriters.SpaceSeparatedTxtWriter(configuration)
Bases: robot.writer.filewriters._DataFileWriter
write(datafile)
```

```python
class robot.writer.filewriters.PipeSeparatedTxtWriter(configuration)
Bases: robot.writer.filewriters._DataFileWriter
write(datafile)
```

```python
class robot.writer.filewriters.TsvFileWriter(configuration)
Bases: robot.writer.filewriters._DataFileWriter
write(datafile)
```
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

4.1. robot package 289
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.ExecutionFailed (message, test_timeout=False, keyword_timeout=False, syntax=False, exit=False, continue_on_failure=False, return_value=None)

Bases: robot.errors.RobotError

Used for communicating failures in test execution.

message
timeout
dont_continue
continue_on_failure
can_continue (teardown=False, templated=False, dry_run=False)
get_errors ()
status
args

exception robot.errors.HandlerExecutionFailed (details)

Bases: robot.errors.ExecutionFailed
timeout
continue_on_failure
args
can_continue (teardown=False, templated=False, dry_run=False)
dont_continue
get_errors()
message
status

```
exception robot.errors.ExecutionFailures(errors, message=None)
   Bases: robot.errors.ExecutionFailed

message
get_errors()
args
can_continue (teardown=False, templated=False, dry_run=False)
continue_on_failure
dont_continue
status
timeout

exception robot.errors.UserKeywordExecutionFailed(run_errors=None, tear-
down_errors=None)
   Bases: robot.errors.ExecutionFailures

message
args
can_continue (teardown=False, templated=False, dry_run=False)
continue_on_failure
dont_continue
get_errors()
status
timeout

exception robot.errors.ExecutionPassed(message=None, **kwargs)
   Bases: robot.errors.ExecutionFailed

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 continuar
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 continuar
earlier_failures
get_errors()
message
set_earlier_failures(failures)
status

timeout

exception  robot.errors.ReturnFromKeyword(\texttt{return\_value})
Bases: robot.errors.ExecutionPassed

Used by ‘Return From Keyword’ keyword.

args
can\_continue (\texttt{teardown=False, templated=False, dry\_run=False})
continue\_on\_failure
dont\_continue
earlier\_failures
get\_errors()
message
set\_earlier\_failures (\texttt{failures})
status
timeout

exception  robot.errors.RemoteError (\texttt{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:

\begin{verbatim}
python -m robot.libdoc
python path/to/robot/libdoc.py
\end{verbatim}

Instead of \texttt{python} it is possible to use also other Python interpreters.

This module also provides \texttt{libdoc()} and \texttt{libdoc\_cli()} functions that can be used programmatically. Other code is for internal usage.

Libdoc itself is implemented in the \texttt{libdocpkg} package.

class  robot.libdoc.LibDoc
Bases: robot.utils.application.Application

\texttt{validate (options, arguments)}

\texttt{main (args, name=", version=", format=None, docformat=None)}

\texttt{console (msg)}

\texttt{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:

```python
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:

```python
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, exit=True)
    Command line execution entry point for post-processing outputs.
        Parameters
            • arguments – Command line options and arguments as a list of strings.
            • 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(*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:

```python
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 programatically. Other code is for internal usage.

```python
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, exit=True)
```

Command line execution entry point for running tests.

Parameters

- `arguments` – Command line options and arguments as a list of strings.
- `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`, `--escape`, `--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 `prerobotmodifier` options allow passing values as Python objects in addition to module names these command line options support. For example, `run('tests', listener=MyListener())`.  

4.1. robot package 297
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:

```python
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.

```python
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:
    ```python
    from robot.testdoc import testdoc_cli
testdoc_cli(['--title', 'Test Plan', 'mytests', 'plan.html'])
    ```

robot.testdoc.testdoc(*arguments, **options)
    Executes Testdoc programatically.

    Arguments and options have same semantics, and options have same names, as arguments and options to Testdoc.

    Example:
    ```python
    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 programatically. 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.

```python
inplace(*paths)
```

Tidy file(s) in-place.

**Parameters**

- **paths** – Paths of the files to to process.

```python
directory(path)
```

Tidy a directory.

**Parameters**

- **path** – Path of the directory to process.

All files in a directory, recursively, are processed in-place.

```python
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.

```python
main(arguments, recursive=False, inplace=False, format='txt', usepipes=False, spacecount=4, lineseparator='
')
```

**validate**(opts, args)

```python
console(msg)
```

```python
execute(*arguments, **options)
```

```python
execute_cli(cli_arguments, exit=True)
```

```python
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
- `DataError` when parsing fails

```python
class robot.tidy.ArgumentValidator
```

Bases: `object`

```python
mode_and_arguments(args, recursive, inplace, **others)
```

```python
format(args, format, inplace, recursive, **others)
```

```python
line_sep(lineseparator, **others)
```

```python
spacecount(spacecount)
```

```python
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, 14
robot.conf.gatherfailed, 15
robot.conf.settings, 16
robot.errors, 289
robot.htmldata, 18
robot.htmldata.htmlfilewriter, 18
robot.htmldata.jartemplate, 19
robot.htmldata.jsonwriter, 19
robot.htmldata.normaltemplate, 20
robot.htmldata.template, 20
robot.jarrunner, 293
robot.libdoc, 293
robot.libdocpkg, 20
robot.libdocpkg.builder, 20
robot.libdocpkg.consoleviewer, 20
robot.libdocpkg.htmlwriter, 21
robot.libdocpkg.javabuilder, 21
robot.libdocpkg.model, 21
robot.libdocpkg.output, 22
robot.libdocpkg.robotbuilder, 22
robot.libdocpkg.specbuilder, 22
robot.libdocpkg.writer, 22
robot.libdocpkg.xmlwriter, 22
robot.libraries, 23
robot.libraries.BuiltIn, 23
robot.libraries.Collections, 45
robot.libraries.DateTime, 51
robot.libraries.Dialogs, 56
robot.librariesdialogs_ipy, 103
robot.librariesdialogs_jy, 103
robot.librariesdialogs_py, 103
robot.libraries.Easter, 57
robot.libraries.OperatingSystem, 57
robot.libraries.Process, 67
robot.libraries.Remote, 73
robot.libraries.Reserved, 74
robot.libraries.Screenshot, 74
robot.libraries.String, 76
robot.libraries.Telnet, 81
robot.libraries.XML, 91
robot.model, 155
robot.model.configurer, 155
robot.model.criticality, 156
robot.model.filter, 156
robot.model.imports, 158
robot.model.itemlist, 159
robot.model.keyword, 159
robot.model.message, 161
robot.model.metadata, 162
robot.model.modelobject, 163
robot.model.modifier, 163
robot.model.namepatterns, 164
robot.model.statistics, 164
robot.model.stats, 165
robot.model.suitestatistics, 168
robot.model.tags, 168
robot.model.tagsetter, 169
robot.model.tagstatistics, 169
robot.model.testcase, 170
robot.model.testsuite, 172
robot.model.totalstatistics, 174
robot.model.visitor, 175
robot.output, 176
robot.output.console, 177
robot.output.console.dotted, 177
robot.output.console.highlighting, 178
robot.output.console.quiet, 179
robot.output.console.verbose, 179
robot.output.debugfile, 180
robot.output.filelogger, 180
robot.output.librarylogger, 180
robot.output.listenerarguments, 181
robot.output.listenermethods, 182
robot.output.listeners, 182
robot.utils.match, 273
robot.utils.misc, 273
robot.utils.normalizing, 274
robot.utils.ordereddict, 275
robot.utils.platform, 275
robot.utils.recommendations, 275
robot.utils.robotenv, 276
robot.utils.robotinspect, 276
robot.utils.robotio, 276
robot.utils.robotpath, 276
robot.utils.robottime, 277
robot.utils.robottypes, 278
robot.utils.robottypes2, 278
robot.utils.setter, 279
robot.utils.sortable, 279
robot.utils.text, 279
robot.utils.unic, 279
robot.utils.utf8reader, 280
robot.variables, 280
robot.variables.assigner, 280
robot.variables.filesetter, 281
robot.variables.finders, 281
robot.variables.isvar, 282
robot.variables.notfound, 282
robot.variables.replacer, 282
robot.variables.scopes, 282
robot.variables.splitter, 284
robot.variables.store, 284
robot.variables.tablesetter, 284
robot.variables.variables, 285
robot.version, 301
robot.writer, 285
robot.writer.aligners, 286
robot.writer.dataextractor, 286
robot.writer.datafilewriter, 286
robot.writer.filewriters, 287
robot.writer.formatters, 288
robot.writer.htmlformatter, 288
robot.writer.htmltemplate, 289
robot.writer.rowsplitter, 289
### Index

**A**
- `absopath()` (in module `robot.utils.robotpath`), 276
- `AbstractLogger` (class in `robot.output.loggerhelper`), 184
- `AbstractLoggerProxy` (class in `robot.output.loggerhelper`), 185
- `accept_gzip_encoding` (`robot.libraries.Remote.TimeoutTransport` attribute), 74
- `acquire()` (`robot.output.pyloggingconf.RobotHandler` method), 186
- `active` (`robot.running.timeouts.KeywordTimeout` attribute), 243
- `active` (`robot.running.timeouts.TestTimeout` attribute), 243
- `add()` (`robot.model.tags.Tags` method), 168
- `add()` (`robot.parsing.comments.CommentCache` method), 189
- `add()` (`robot.parsing.comments.Comments` method), 189
- `add()` (`robot.parsing.model.KeywordTable` method), 195
- `add()` (`robot.parsing.model.TestCaseTable` method), 195
- `add()` (`robot.parsing.model.VariableTable` method), 194
- `add()` (`robot.parsing.populators.FromFilePopulator` method), 197
- `add()` (`robot.parsing.settings.ImportList` method), 202
- `add()` (`robot.parsing.settings.MetadataList` method), 202
- `add()` (`robot.parsing.tablepopulators.DocumentationPopulator` method), 203
- `add()` (`robot.parsing.tablepopulators.ForLoopPopulator` method), 203
- `add()` (`robot.parsing.tablepopulators.KeywordTablePopulator` method), 203
- `add()` (`robot.parsing.tablepopulators.MetadataPopulator` method), 204
- `add()` (`robot.parsing.tablepopulators.NullPopulator` method), 202
- `add()` (`robot.parsing.tablepopulators.Populator` method), 202
- `add()` (`robot.parsing.tablepopulators.SettingPopulator` method), 203
- `add()` (`robot.parsing.tablepopulators.SettingTablePopulator` method), 203
- `add()` (`robot.parsing.tablepopulators.StepPopulator` method), 203
- `add()` (`robot.parsing.tablepopulators.TestCasePopulator` method), 203
- `add()` (`robot.parsing.tablepopulators.TestTablePopulator` method), 203
- `add()` (`robot.parsing.tablepopulators.UserKeywordPopulator` method), 203
- `add()` (`robot.parsing.tablepopulators.VariablePopulator` method), 203
- `add()` (`robot.parsing.tablepopulators.VariableTablePopulator` method), 203
- `add()` (`robot.parsing.tablepopulators.StepPopulator` method), 203
- `add()` (`robot.parsing.tablepopulators.TestCasePopulator` method), 203
- `add()` (`robot.parsing.tablepopulators.TestTablePopulator` method), 203
- `add()` (`robot.parsing.tablepopulators.UserKeywordPopulator` method), 203
- `add()` (`robot.parsing.tablepopulators.VariablePopulator` method), 203
- `add()` (`robot.parsing.tablepopulators.VariableTablePopulator` method), 203
- `add()` (`robot.parsing.tablepopulators.StepPopulator` method), 203
- `add()` (`robot.parsing.tablepopulators_TestCasePopulator` method), 203
- `add()` (`robot.parsing.tablepopulators_TestCaseTablePopulator` method), 203
- `add()` (`robot.parsing.tablepopulators_TestCasePopulator` method), 203
- `add()` (`robot.parsing.tablepopulators_TestCaseTablePopulator` method), 203
- `add()` (`robot.parsing.tablepopulators_TestCasePopulator` method), 203
- `add()` (`robot.parsing.tablepopulators_TestCaseTablePopulator` method), 203
- `add()` (`robot.parsing.tablepopulators_TestCasePopulator` method), 203
- `add()` (`robot.parsing.tablepopulators_TestCaseTablePopulator` method), 203
- `add()` (`robot.parsing.tablepopulators_TestCaseTablePopulator` method), 203
add_for_loop() (robot.parsing.model.UserKeyword method), 196
add_library() (robot.parsing.model.InitFileSettingTable method), 194
add_library() (robot.parsing.model.ResourceFileSettingTable method), 193
add_library() (robot.parsing.model.TestCaseFileSettingTable method), 193
add_metadata() (robot.parsing.model.InitFileSettingTable method), 194
add_metadata() (robot.parsing.model.ResourceFileSettingTable method), 193
add_metadata() (robot.parsing.model.TestCaseFileSettingTable method), 193
add_name() (robot.parsing.restsupport.CaptureRobotData method), 197
add_path() (in module robot.pythonpathsetter), 294
add_resource() (robot.parsing.model.InitFileSettingTable method), 194
add_resource() (robot.parsing.model.ResourceFileSettingTable method), 193
add_resource() (robot.parsing.model.TestCaseFileSettingTable method), 193
add_result() (robot.result.executionresult.CombinedResult method), 214
add_stat() (robot.model.stats.SuiteStat method), 166
add_step() (robot.parsing.model.ForLoop method), 196
add_step() (robot.parsing.model.TestCase method), 195
add_step() (robot.parsing.model.UserKeyword method), 196
add_tags (robot.model.configurer.SuiteConfigurer attribute), 155
add_tags (robot.result.configurer.SuiteConfigurer attribute), 212
add_test() (robot.model.stats.CombinedTagStat method), 167
add_test() (robot.model.stats.CriticalTagStat method), 167
add_test() (robot.model.stats.Stat method), 166
add_test() (robot.model.stats.TagStat method), 167
add_test() (robot.model.stats.TotalStat method), 166
add_test() (robot.model.suitestatistics.SuiteStatisticsBuilder method), 168
add_test() (robot.model.tagstatistics.TagStatisticsBuilder method), 170
add_test() (robot.model.totalstatistics.TotalStatisticsBuilder method), 174
add_time_to_date() (in module robot.libraries.DateTime), 55
add_time_to_time() (in module robot.libraries.DateTime), 56
add_variables() (robot.parsing.model.InitFileSettingTable method), 194
add_variables() (robot.parsing.model.ResourceFileSettingTable method), 193
add_variables() (robot.parsing.model.TestCaseFileSettingTable method), 193
addFilter() (robot.output.pyloggingconf.RobotHandler method), 186
after() (robot.libraries.dialogs_py.InputDialog method), 116
after() (robot.libraries.dialogs_py.MessageDialog method), 103
after() (robot.libraries.dialogs_py.PassFailDialog method), 142
after() (robot.libraries.dialogs_py.SelectionDialog method), 129
after_cancel() (robot.libraries.dialogs_py.InputDialog method), 116
after_cancel() (robot.libraries.dialogs_py.MessageDialog method), 103
after_cancel() (robot.libraries.dialogs_py.PassFailDialog method), 142
after_cancel() (robot.libraries.dialogs_py.SelectionDialog method), 129
after_idle() (robot.libraries.dialogs_py.InputDialog method), 116
after_idle() (robot.libraries.dialogs_py.MessageDialog method), 103
after_idle() (robot.libraries.dialogs_py.PassFailDialog method), 142
after_idle() (robot.libraries.dialogs_py.SelectionDialog method), 129
align_row() (robot.writer.aligners.ColumnAligner method), 286
align_row() (robot.writer.aligners.FirstColumnAligner method), 286
align_row() (robot.writer.aligners.NullAligner method), 286
align_rows() (robot.writer.aligners.ColumnAligner method), 286
align_rows() (robot.writer.aligners.FirstColumnAligner method), 286
align_rows() (robot.writer.aligners.NullAligner method), 286
align_rows() (robot.writer.aligners.NullAligner method), 286
align_all() (robot.model.keyword.Keywords attribute), 161
all (robot.model.totalstatistics.TotalStatistics attribute), 174
all (robot.parsing.datarow.DataRow attribute), 189
also_teardown_message (robot.running.status.ParentMessage attribute), 259
also_teardown_message (robot.running.status.SuiteMessage attribute), 259
ALLOWED_TYPES (robot.model.imports.Import attribute), 158
Index
clear() (robot.variables.scopes.GlobalVariables method), 283
clear() (robot.variables.store.VariableStore method), 284
clear() (robot.variables.variables.Variables method), 285
clear_cdata_mode() (robot.parsing.htmlreader.HtmlReader method), 190
clear_element() (robot.libraries.XML.XML method), 101
client() (robot.libraries.dialogs_py.InputDialog method), 117
client() (robot.libraries.dialogs_py.MessageDialog method), 104
client() (robot.libraries.dialogs_py.PassFailDialog method), 143
client() (robot.libraries.dialogs_py.SelectionDialog method), 130
clipboard_append() (robot.libraries.dialogs_py.InputDialog method), 117
clipboard_append() (robot.libraries.dialogs_py.MessageDialog method), 104
clipboard_append() (robot.libraries.dialogs_py.PassFailDialog method), 143
clipboard_append() (robot.libraries.dialogs_py.SelectionDialog method), 130
clipboard_clear() (robot.libraries.dialogs_py.InputDialog method), 118
clipboard_clear() (robot.libraries.dialogs_py.MessageDialog method), 105
clipboard_clear() (robot.libraries.dialogs_py.PassFailDialog method), 144
clipboard_clear() (robot.libraries.dialogs_py.SelectionDialog method), 131
clipboard_get() (robot.libraries.dialogs_py.InputDialog method), 118
clipboard_get() (robot.libraries.dialogs_py.MessageDialog method), 105
clipboard_get() (robot.libraries.dialogs_py.PassFailDialog method), 144
clipboard_get() (robot.libraries.dialogs_py.SelectionDialog method), 131
close() (robot.libraries.Remote.TimeoutTransport method), 74
close() (robot.libraries.Telnet.TelnetConnection method), 89
close() (robot.output.filelogger.FileLogger method), 180
close() (robot.output.logger.Logger method), 183
close() (robot.output.output.Output method), 185
close() (robot.output.pyloggingconf.RobotHandler method), 186
close() (robot.output.xmllogger.XmlLogger method), 187
close() (robot.parsing.htmlreader.HtmlReader method), 190
close() (robot.reporting.outputwriter.OutputWriter method), 206
close() (robot.utils.application.DefaultLogger method), 262
close() (robot.utils.markupwriters.HtmlWriter method), 272
close() (robot.utils.markupwriters.NullMarkupWriter method), 273
close() (robot.utils.markupwriters.XmlWriter method), 273
close_all() (robot.utils.connectioncache.ConnectionCache method), 267
close_all_connections() (robot.libraries.Telnet.TelnetConnection method), 85
close_connection() (robot.libraries.Telnet.TelnetConnection method), 87
close_global_library_listeners() (robot.running.importer.Importer method), 246
close_streams() (robot.libraries.Process.ExecutionResult method), 73
cmdline2list() (in module robot.utils.argumentparser), 263
coerce() (robot.libraries.Remote.ArgumentCoercer method), 73
coerce() (robot.running.arguments.javaargumentcoercer.BooleanCoercer method), 242
coerce() (robot.running.arguments.javaargumentcoercer.FloatCoercer method), 242
coerce() (robot.running.arguments.javaargumentcoercer.IntegerCoercer method), 242
coerce() (robot.running.arguments.javaargumentcoercer.JavaArgumentCoercer method), 242
coerce() (robot.running.arguments.javaargumentcoercer.NullCoercer method), 242
CoercerFinder (class in robot.running.arguments.javaargumentcoercer), 242
Collections (class in robot.libraries.Collections), 45
colormapwindows() (robot.libraries.dialogs_py.InputDialog method), 118
colormapwindows() (robot.libraries.dialogs_py.MessageDialog method), 105
colormapwindows() (robot.libraries.dialogs_py.PassFailDialog method), 144
colormapwindows() (robot.libraries.dialogs_py.SelectionDialog method), 131
colormodel() (robot.libraries.dialogs_py.InputDialog method), 118
colormodel() (robot.libraries.dialogs_py.MessageDialog method), 105
colormodel() (robot.libraries.dialogs_py.PassFailDialog method), 144
colormodel() (robot.libraries.dialogs_py.SelectionDialog method), 131
ColumnAligner (class in robot.writer.aligners), 286
columnconfigure() (robot.libraries.dialogs_py.InputDialog method), 117
continue_on_failure (robot.errors.HandlerExecutionFailed attribute), 290
continue_on_failure (robot.errors.PassExecution attribute), 292
continue_on_failure (robot.errors.ReturnFromKeyword attribute), 293
continue_on_failure (robot.errors.UserKeywordExecutionFailed attribute), 291
ContinueForLoop, 292
convert() (robot.libdocpkg.htmlwriter.JsonConverter method), 21
convert() (robot.testdoc.JsonConverter method), 299
convert_date() (in module robot.libraries.DateTime), 54
convert_time() (in module robot.libraries.DateTime), 55
convert_to_binary() (robot.libraries.BuiltIn.Builtin method), 25
convert_to_boolean() (robot.libraries.BuiltIn.Builtin method), 25
convert_to_bytes() (robot.libraries.BuiltIn.Builtin method), 25
convert_to_dictionary() (robot.libraries.Collections.Collections method), 47
convert_to_hex() (robot.libraries.BuiltIn.Builtin method), 25
convert_to_integer() (robot.libraries.BuiltIn.Builtin method), 26
convert_to_list() (robot.libraries.Collections.Collections method), 47
convert_to_lowercase() (robot.libraries.String.String method), 76
convert_to_number() (robot.libraries.BuiltIn.Builtin method), 26
convert_to_octal() (robot.libraries.BuiltIn.Builtin method), 26
convert_to_string() (robot.libraries.BuiltIn.Builtin method), 26
convert_to_uppercase() (robot.libraries.String.String method), 76
copy() (robot.model.keyword.Keyword method), 160
copy() (robot.model.message.Message method), 161
copy() (robot.model.metadata.Metadata method), 162
copy() (robot.model.modelobject.ModelObject method), 163
copy() (robot.model.testcase.TestCase method), 171
copy() (robot.model.testsuite.TestSuite method), 173
copy() (robot.output.loggerhelper.Message method), 184
copy() (robot.parsing.model.ForLoop method), 196
copy() (robot.parsing.model.TestCase method), 195
copy() (robot.parsing.model.UserKeyword method), 196
copy() (robot.result.model.Keyword method), 225
copy() (robot.result.model.Message method), 223
copy() (robot.result.model.TestCase method), 226
copy() (robot.result.model.TestSuite method), 228
copy() (robot.running.model.ForLoop method), 250
copy() (robot.running.model.Keyword method), 248
copy() (robot.running.model.TestCase method), 251
copy() (robot.running.model.TestSuite method), 253
copy() (robot.utils.dotdict.DotDict method), 267
copy() (robot.utils.normalizing.NormalizedDict method), 274
copy() (robot.variables.scopes.GlobalVariables method), 283
copy() (robot.variables.variables.Variables method), 285
copy_dictionary() (robot.libraries.Collections.Collections method), 47
copy_directory() (robot.libraries.OperatingSystem.OperatingSystem method), 63
copy_element() (robot.libraries.XML.XML method), 101
copy_files() (robot.libraries.OperatingSystem.OperatingSystem method), 63
copy_list() (robot.libraries.Collections.Collections method), 47
count_items_in_directory() (robot.libraries.OperatingSystem.OperatingSystem method), 66
count_values_in_list() (robot.libraries.Collections.Collections method), 47
create() (robot.model.imports.Imports method), 158
create() (robot.model.itemlist.ItemList method), 159
create() (robot.model.keyword.Keywords method), 161
create() (robot.model.message.Messages method), 162
create() (robot.model.testcase.TestCases method), 171
create() (robot.model.testsuite.TestSuites method), 173
create_dictionary() (robot.libraries.BuiltIn.Builtin method), 27
create_directory() (robot.libraries.OperatingSystem.OperatingSystem method), 62
create_link_target() (robot.reporting.jsbuildingcontext.JsBuildingContext method), 205
create_list() (robot.libraries.BuiltIn.Builtin method), 27
create_runner() (robot.running.handlers.EmbeddedArgumentsHandler method), 246
create_runner() (robot.running.handlerstore.HandlerStore method), 246
create_runner() (robot.running.usererrorhandler.UserErrorHandler method), 246

316
method), 260
create_runner() (robot.running.userkeyword.EmbeddedArgumentsHandler method), 261
create_runner() (robot.running.userkeyword.UserKeywordHandler method), 261
createLock() (robot.output.pyloggingconf.RobotHandler method), 186
critical (robot.model.stats.TagStat attribute), 167
critical (robot.model.tagstatistics.TagStatistics attribute), 169
critical (robot.model.totalstatistics.TotalStatistics attribute), 174
critical (robot.result.model.TestCase attribute), 226
critical_failure_occurred() (robot.running.status.SuiteStatus method), 258
critical_failure_occurred() (robot.running.status.TestStatus method), 258
critical_tags (robot.conf.settings.RebotSettings attribute), 17
critical_tags (robot.conf.settings.RobotSettings attribute), 17
CriticalTagStat (class in robot.model.stats), 167
CssFileWriter (class in robot.htmldata.htmlfilewriter), 19
current (robot.model.suitestatistics.SuiteStatisticsBuilder attribute), 168
current (robot.running.context.ExecutionContexts attribute), 245
current (robot.utils.connectioncache.ConnectionCache attribute), 266
current (robot.variables.scopes.VariableScopes attribute), 282
current_index (robot.utils.connectioncache.ConnectionCache attribute), 267
current_output (robot.libaries.Telnet.TerminalEmulator attribute), 91
cut_long_message() (in module robot.utils.text), 279
data (robot.parsing.datarow.DataRow attribute), 189
DataError, 289
DataExtractor (class in robot.writer.dataextractor), 286
DataFileWriter (class in robot.writer.datafilewriter), 286
DataRow (class in robot.parsing.datarow), 189
debug() (in module robot.api.logger), 14
debug() (in module robot.output.librarylogger), 180
debug() (robot.output.filelogger.FileLogger method), 180
debug() (robot.output.logger.Logger method), 183
debug() (robot.output.loggerhelper.AbstractLogger method), 184
debug() (robot.output.output.Output method), 185
debug() (robot.parsing.restsupport.CaptureRobotData method), 197
debug_file (robot.conf.settings.RobotSettings attribute), 16
DebugFile() (in module robot.output.debugfile), 180
decode_bytes_to_string() (robot.libaries.String.String method), 77
dedent() (robot.parsing.datarow.DataRow method), 190
deeprcopy() (robot.model.keyword.Keyword method), 160
deeprcopy() (robot.model.message.Message method), 162
deeprcopy() (robot.model.modelobject.ModelObject method), 163
deeprcopy() (robot.model.testcase.TestCase method), 171
deeprcopy() (robot.model.testsuite.TestSuite method), 173
deeprcopy() (robot.output.loggerhelper.Message method), 184
deeprcopy() (robot.result.model.Keyword method), 225
deeprcopy() (robot.result.model.Message method), 223
deeprcopy() (robot.result.model.TestCase method), 226
deeprcopy() (robot.result.model.TestSuite method), 228
deeprcopy() (robot.running.model.ForLoop method), 250
deeprcopy() (robot.running.model.Keyword method), 249
deeprcopy() (robot.running.model.TestCase method), 251
deeprcopy() (robot.running.model.TestSuite method), 253
DefaultLogger (class in robot.utils.application), 262
DefaultValue (class in robot.running.arguments.argumentmapper), 240
deiconify() (robot.libaries.dialogs_py.InputDialog method), 118
deiconify() (robot.libaries.dialogs_py.MessageDialog method), 105
deiconify() (robot.libaries.dialogs_py.PassFailDialog method), 144
deiconify() (robot.libaries.dialogs_py.SelectionDialog method), 131
del_env_var() (in module robot.utils.robotenv), 276
deletecommand() (robot.libaries.dialogs_py.MessageDialog method), 118
deletecommand() (robot.libaries.dialogs_py.MessageDialog method), 105
deletecommand() (robot.libaries.dialogs_py.PassFailDialog method), 144
deletecommand() (robot.libaries.dialogs_py.SelectionDialog method), 131
denominator (robot.reporting.stringcache.StringIndex attribute), 208
destroy() (robot.libaries.dialogs_py.InputDialog method), 118
destroy() (robot.libaries.dialogs_py.MessageDialog method), 105
destroy() (robot.libaries.dialogs_py.PassFailDialog method), 144
Index 317
method), 131
DictDumper (class in robot.htmldata.jsonwriter), 19
dictionaries_should_be_equal()
(robot.libraries.Collections.Collections method), 47
dictionary_should_contain_item()
(robot.libraries.Collections.Collections method), 47
dictionary_should_contain_key()
(robot.libraries.Collections.Collections method), 47
dictionary_should_contain_sub_dictionary()
(robot.libraries.Collections.Collections method), 48
dictionary_should_contain_value()
(robot.libraries.Collections.Collections method), 48
dictionary_should_not_contain_key()
(robot.libraries.Collections.Collections method), 48
dictionary_should_not_contain_value()
(robot.libraries.Collections.Collections method), 48
DictToKwargs (class in
robot.running.arguments.argumentresolver), 241
.DictVariableTableValue (class in
robot.variables.tablesetter), 285
directive_error() (robot.parsing.restsupport.CaptureRobotData method), 197
directory (robot.model.imports.Import attribute), 158
directory (robot.parsing.model.InitFileSettingTable attribute), 194
directory (robot.parsing.model.KeywordTable attribute), 195
directory (robot.parsing.model.ResourceFileSettingTable attribute), 193
directory (robot.parsing.model_TestCase attribute), 195
directory (robot.parsing.model_TestCaseFileSettingTable attribute), 193
directory (robot.parsing.model_TestCaseTable attribute), 195
directory (robot.parsing.model_TestCase attribute), 196
directory (robot.parsing.model_VariableTable attribute), 194
directory (robot.parsing.settings.Arguments attribute), 200
directory (robot.parsing.settings.Documentation attribute), 199
directory (robot.parsing.settings_Fixture attribute), 199
directory (robot.parsing.settings.Library attribute), 201
directory (robot.parsing.settings_MetaData attribute), 201
directory (robot.parsing.settings_Resource attribute), 201
directory (robot.parsing.settings_Return attribute), 200
directory (robot.parsing.settings_Setting attribute), 198
directory (robot.parsing.settings_Tags attribute), 200
directory (robot.parsing.settings_Template attribute), 199
directory (robot.parsing.settings Timeout attribute), 200
directory (robot.parsing.settings_Variables attribute), 202
directory() (robot.tidy.Tidy method), 300
directory_should_be_empty()
directory_should_exist() (robot.libraries.OperatingSystem.OperatingSystem method), 60
directory_should_not_be_empty()
directory_should_not_exist()
(robot.libraries.OperatingSystem.OperatingSystem method), 60
disable_curdir_processing() (in module robot.parsing), 189
disable_library_import_logging()
(robot.output.logger.Logger method), 183
disable_message_cache()
(robot.output.logger.Logger method), 183
discard_suite_scope() (robot.output.listeners.LibraryListeners method), 182
discard_suite_scope() (robot.output.listeners.LibraryListener method), 182
doc (robot.model.keyword.Keyword attribute), 159
doc (robot.model.stats.TagStat attribute), 166
doc (robot.model_TestCase attribute), 170
doc (robot.model_TestCaseSuite attribute), 172
doc (robot.result.model_TestCase attribute), 225
doc (robot.result.model_TestCase attribute), 226
doc (robot.result.model_TestCase attribute), 229
doc (robot.running.model_ForLoop attribute), 250
doc (robot.running.model_TestCase attribute), 249
doc (robot.running.model_TestCase attribute), 251
doc (robot.running.model_TestCase attribute), 254
doc (robot.running._UserErrorHandler.UserErrorHandler attribute), 260
doc_format (robot.libdocpkg.model.LibraryDoc attribute), 21
DocFormatter (class in robot.libdocpkg.htmlwriter), 21
DocHandler (class in robot.result.xmlelementhandlers), 236
DocToHtml (class in robot.libdocpkg.htmlwriter), 21
Documentation (class in robot.parsing.settings), 198
DocumentationBuilder() (in module robot.libdocpkg.builder), 20
DocumentationCell (class in robot.writer.htmlformatter), 288
DocumentationPopulator (class in robot.parsing.tablepopulators), 203
dont_continue (robot.errors.ContinueForLoop attribute), 292
don't_continue (robot.errors.ExecutionFailed attribute), 290
don't_continue (robot.errors.ExecutionFailures attribute), 291
don't_continue (robot.errors.ExecutionPassed attribute), 291
don't_continue (robot.errors.ExitForLoop attribute), 292
don't_continue (robot.errors.HandlerExecutionFailed attribute), 290
don't_continue (robot.errors.PassExecution attribute), 292
don't_continue (robot.errors.ReturnFromKeyword attribute), 293
don't_continue (robot.errors.UserKeywordExecutionFailed attribute), 291
DosHighlighter (class in robot.output.console.highlighting), 178
DotDict (class in robot.utils.dotdict), 267
DottedImporter (class in robot.utils.importer), 272
DottedOutput (class in robot.output.console.dotted), 177
dry_run (robot.conf.settings.RobotSettings attribute), 16
dry_run() (robot.running.librarykeywordrunner.EmbeddedArgumentsRunner method), 247
dry_run() (robot.running.librarykeywordrunner.LibraryKeywordRunner method), 247
dry_run() (robot.running.librarykeywordrunner.RunKeywordRunner method), 247
dry_run() (robot.running.usererrorhandler.UserErrorHandler method), 261
dry_run() (robot.running.userkeywordrunner.EmbeddedArgumentsRunner method), 261
dry_run() (robot.running.userkeywordrunner.UserKeywordRunner method), 261
dump() (robot.htmldata.jsonwriter.DictDumper method), 19
dump() (robot.htmldata.jsonwriter.IntegerDumper method), 19
dump() (robot.htmldata.jsonwriter.JsonDumper method), 19
dump() (robot.htmldata.jsonwriter.MappingDumper method), 20
dump() (robot.htmldata.jsonwriter.NoneDumper method), 20
dump() (robot.htmldata.jsonwriter.StringDumper method), 19
dump() (robot.htmldata.jsonwriter.TupleListDumper method), 19
dump() (robot.reporting.stringcache.StringCache method), 209
DynamicArgumentParser (class in robot.running.arguments.embedded), 242
DynamicHandler() (in module robot.running.handlers), 246
earlier_failures (robot.errors.ContinueForLoop attribute), 292
earlier_failures (robot.errors.ExecutionPassed attribute), 291
earlier_failures (robot.errors.ExitForLoop attribute), 292
earlier_failures (robot.errors.PassExecution attribute), 292
earlier_failures (robot.errors.ReturnFromKeyword attribute), 293
elapsed (robot.model.stats.Stat attribute), 166
elapsed (robot.model.stats.SuiteStat attribute), 166
elapsed_time_to_string() (in module robot.utils.robottime), 278
elapsedtime (robot.result.model.Keyword attribute), 224
elapsedtime (robot.result.model.TestCase attribute), 226
elapsedtime (robot.result.model.TestSuite attribute), 228
element() (robot.utils.markupwriters.HtmlWriter method), 272
element() (robot.utils.markupwriters.NullMarkupWriter method), 273
element() (robot.utils.markupwriters.XmlWriter method), 273
element_attribute_should_be() (robot.libraries.XML.XML method), 98
element_attribute_should_match() (robot.libraries.XML.XML method), 98
element_should_exist() (robot.libraries.XML.XML method), 96
element_should_not_exist() (robot.libraries.XML.XML method), 96
element_should_not_have_attribute() (robot.libraries.XML.XML method), 98
element_text_should_be() (robot.libraries.XML.XML method), 97
element_text_should_match() (robot.libraries.XML.XML method), 97
element_to_string() (robot.libraries.XML.XML method), 101
ElementComparator (class in robot.libraries.XML), 102
ElementFinder (class in robot.libraries.XML), 102
elements_should_be_equal() (robot.libraries.XML.XML method), 98
elements_should_match() (robot.libraries.XML.XML method), 99
EmbeddedArgumentParser (class in robot.running.arguments.embedded), 242
EmbeddedArguments (class in robot.running.arguments.embedded), 242
EmbeddedArgumentsHandler (class in robot.running.handlers), 246
end_message() (robot.result.suitetearndownfailed.SuiteTeardownFailed method), 232
end_message() (robot.result.suitetearndownfailed.SuiteTeardownFailureHandler method), 231
end_message() (robot.result.visitor.ResultVisitor method), 234
end_message() (robot.running.randomizer.Randomizer method), 256
end_message() (robot.running.runner.Runner method), 257
end_result() (robot.output.xmllogger.XmlLogger method), 188
end_result() (robot.result.visitor.ResultVisitor method), 233
end_result() (robot.running.randomizer.Randomizer method), 256
end_result() (robot.running.runner.Runner method), 257
end_statistics() (robot.output.xmllogger.XmlLogger method), 187
end_statistics() (robot.result.visitor.ResultVisitor method), 233
end_splitting() (robot.reporting.jsbuildingcontext.JsBuildingContext method), 205
end_stat() (robot.output.xmllogger.XmlLogger method), 188
end_stat() (robot.result.visitor.ResultVisitor method), 234
end_suite() (robot.conf.gatherfailed.GatherFailedSuites method), 15
end_suite() (robot.conf.gatherfailed.GatherFailedTests method), 16
end_suite() (robot.model.configurer.SuiteConfigurer method), 155
end_suite() (robot.model.filter.EmptySuiteRemover method), 156
end_suite() (robot.model.filter.Filter method), 157
end_suite() (robot.model.modifier.ModelModifier method), 163
end_suite() (robot.model.statistics.StatisticsBuilder method), 165
end_suite() (robot.model.suitestatistics.SuiteStatisticsBuilder method), 168
end_suite() (robot.model.tagsetter.TagSetter method), 169
end_suite() (robot.model.totalstatistics>TotalStatisticsBuilder method), 174
end_suite() (robot.result.configurer.SuiteConfigurer method), 212
end_suite() (robot.result.keywordremover.AllKeywordsRemover method), 215
end_suite() (robot.result.keywordremover.ByNameKeywordRemover method), 217
end_suite() (robot.result.keywordremover.ByTagKeywordRemover method), 218
end_suite() (robot.result.keywordremover.ForLoopItemsRemover method), 219
end_suite() (robot.result.keywordremover.PassedKeywordRemover method), 216
end_suite() (robot.result.keywordremover.WaitUntilKeywordSucceedsRemover method), 220
end_suite() (robot.result.keywordremover.WarningAndErrorFinder method), 221
end_suite() (robot.result.messagefilter.MessageFilter method), 222
end_suite() (robot.result.resultbuilder.RemoveKeywords method), 231
end_suite() (robot.result.suitetearndownfailed.SuiteTeardownFailed method), 232
end_suite() (robot.result.suitetearndownfailed.SuiteTeardownFailureHandler method), 231
end_suite() (robot.running.context.ExecutionContexts method), 245
end_suite() (robot.running.libraryscopes.GlobalScope method), 247
end_suite() (robot.running.libraryscopes.TestCaseScope method), 248
end_suite() (robot.running.libraryscopes.TestSuiteScope method), 247
end_suite() (robot.running.namespace.Namespace method), 322

Index
Index 323
EndKeywordArguments (class in robot.output.listenerarguments), 181
EndSuiteArguments (class in robot.output.listenerarguments), 181
EndTestArguments (class in robot.output.listenerarguments), 181
endtime (robot.result.model.Keyword attribute), 224
detime (robot.result.model.TestCase attribute), 226
detime (robot.result.model.TestSuite attribute), 227
entitydefs (robot.parsing.htmlreader.HtmlReader attribute), 190
evaluation_variable_should_be_set() (robot.libraries.OperatingSystem.OperatingSystem method), 64
evaluation_variable_should_not_be_set() (robot.libraries.OperatingSystem.OperatingSystem method), 64
EnvironmentFinder (class in robot.variables.finders), 281
eof() (robot.parsing.populators.FromFilePopulator method), 197
eq() (in module robot.utils.match), 273
error() (in module robot.api.logger), 14
error() (robot.output.console.highlighting.HighlightingStreamWriter method), 178
error() (robot.output.console.verbose.VerboseWriter method), 179
error() (robot.output.filelogger.FileLogger method), 180
error() (robot.output.logger.Logger method), 183
error() (robot.output.loggerhelper.AbstractLogger method), 184
error() (robot.output.output.Output method), 185
error() (robot.parsing.htmlreader.HtmlReader method), 191
error() (robot.parsing.restsupport.CaptureRobotData method), 198
error() (robot.utils.application.DefaultLogger method), 262
error_occurred() (robot.running.status.Exit method), 258
error_occurred() (robot.running.status.TestStatus method), 258
error_occurred() (robot.running.status.TestStatus method), 258
ErrorDetails() (in module robot.utils.error), 269
ErrorMessageBuilder (class in robot.reporting.jsmodelbuilders), 206
ErrorsBuilder (class in robot.reporting.jsmodelbuilders), 205
ErrorsHandler (class in robot.result.xmlelementhandlers), 238
escape() (in module robot.utils.escaping), 269
EscapeFinder (class in robot.utils.escaping), 269
ETSource (class in robot.utils.etreewrapper), 269
evaluate() (robot.libraries.Builtin.Builtin method), 27
evaluate_xpath() (robot.libraries.XML.XML method), 102
event_add() (robot.libraries.dialogs_pyInputDialog method), 119
event_add() (robot.libraries.dialogs_pyMessageDialog method), 106
event_add() (robot.libraries.dialogs_pyPassFailDialog method), 145
event_add() (robot.libraries.dialogs_pySelectionDialog method), 132
event_delete() (robot.libraries.dialogs_pyInputDialog method), 119
event_delete() (robot.libraries.dialogs_pyMessageDialog method), 106
event_delete() (robot.libraries.dialogs_pyPassFailDialog method), 145
event_delete() (robot.libraries.dialogs_pySelectionDialog method), 132
event_generate() (robot.libraries.dialogs_pyInputDialog method), 119
event_generate() (robot.libraries.dialogs_pyMessageDialog method), 106
event_generate() (robot.libraries.dialogs_pyPassFailDialog method), 145
event_generate() (robot.libraries.dialogs_pySelectionDialog method), 132
event_info() (robot.libraries.dialogs_pySelectionDialog method), 132
event_info() (robot.libraries.dialogs_pyInputDialog method), 119
event_info() (robot.libraries.dialogs_pyMessageDialog method), 106
execute() (robot.libraries.libdoc.LibDoc method), 293
execute() (robot.rebot.Rebot method), 295
execute() (robot.run.RobotFramework method), 296
execute() (robot.running.timeouts.ironpython.Timeout method), 243
execute() (robot.running.timeouts.jython.Timeout method), 243
execute() (robot.running.timeouts.posix.Timeout method), 244
execute() (robot.running.timeouts.windows.Timeout method), 244
execute() (robot.testdoc.TestDoc method), 298
execute() (robot.tidy.TidyCommandLine method), 300
execute() (robot.utils.application.Application method), 262
execute_cli() (robot.libdoc.LibDoc method), 293
execute_cli() (robot.rebot.Rebot method), 295
execute_cli() (robot.run.RobotFramework method), 296
execute_cli() (robot.testdoc.TestDoc method), 298
execute_cli() (robot.tidy.TidyCommand method), 300
execute_cli() (robot.utils.application.Application method), 262
execute_command() (robot.libraries.Telnet.TelnetConnection method), 89
execute_manual_step() (in module robot.libraries.Dialogs), 56
ExecutionContexts (class in robot.running.context), 245
ExecutionErrors (class in robot.result.executionerrors), 212
ExecutionFailed, 290
ExecutionFailures, 291
ExecutionPassed, 291
ExecutionResult (class in robot.libraries.Process), 72
ExecutionResult() (in module robot.result.resultbuilder), 230
ExecutionResultBuilder (class in robot.result.resultbuilder), 230
Exit (class in robot.running.status), 258
exit_for_loop() (robot.libraries.BuiltIn.BuiltIn method), 27
exit_for_loop_if() (robot.libraries.BuiltIn.BuiltIn method), 27
exit_on_error (robot.conf.settings.RobotSettings attribute), 16
exit_on_error_message (robot.running.status.TestMessage attribute), 259
exit_on_failure (robot.conf.settings.RobotSettings attribute), 16
exit_on_failure_message (robot.running.status.TestMessage attribute), 259
exit_on_fatal_message (robot.running.status.TestMessage attribute), 259
ExitForLoop, 292
expect() (robot.libraries.Telnet.TelnetConnection method), 89
extend() (robot.model.imports.Imports method), 158
extend() (robot.model.itemlist.ItemList method), 159
extend() (robot.model.keyword.Keywords method), 161
extend() (robot.model.message.Messages method), 162
extend() (robot.model.testcase.TestCases method), 171
extend() (robot.model.testsuite.TestSuites method), 173
ExtendedFinder (class in robot.variables.finders), 281
fail() (in module robot.utils.asserts), 265
fail() (robot.libraries.BuiltIn.BuiltIn method), 27
fail() (robot.output.filelogger.FileLogger method), 180
fail() (robot.output.logger.Logger method), 184
fail() (robot.output.loggerhelper.AbstractLogger method), 184
fail() (robot.output.output.Output method), 185
failed (robot.model.stats.Stat attribute), 166
Failure (class in robot.running.status), 258
failure_occurred() (robot.running.status.Exit method), 258
failures (robot.running.status.SuiteStatus attribute), 258
failures (robot.running.status.TestStatus attribute), 258
fatal_error() (robot.libraries.BuiltIn.BuiltIn method), 28
feed() (robot.libraries.Telnet.TelnetTerminalEmulator method), 91
feed() (robot.parsing.htmlreader.HtmlReader method), 191
fetch_from_left() (robot.libraries.String.StringReader method), 79
fetch_from_right() (robot.libraries.String.String method), 80
file() (robot.tidy.Tidy method), 299
file_should_be_empty() (robot.libraries.OperatingSystem.OperatingSystem method), 61
file_should_exist() (robot.libraries.OperatingSystem.OperatingSystem method), 60
file_should_not_be_empty() (robot.libraries.OperatingSystem.OperatingSystem method), 61
file_should_not_exist() (robot.libraries.OperatingSystem.OperatingSystem method), 60
FileWriter() (in module robot.writer.filewriters), 287
fill_named() (robot.running.arguments.argumentmapper.KeywordCallTemplate method), 240
fill_positional() (robot.running.arguments.argumentmapper.KeywordCallTemplate method), 240
fill_rawq() (robot.libraries.Telnet.TelnetConnection method), 89
Filter (class in robot.model.filter), 157
filter() (robot.model.testsuite.TestSuite method), 173
filter() (robot.output.pyloggingconf.RobotHandler method), 186
filter() (robot.result.model.TestSuite method), 229
filter() (robot.running.model.TestSuite method), 254
filter_messages() (robot.result.model.TestSuite method), 228
final_argument_whitespace (robot.parsing.restsupport.CaptureRobotData attribute), 198
find() (robot.variables.finders.EmptyFinder method), 281
find() (robot.variables.finders.EnvironmentFinder method), 281
find() (robot.variables.finders.ExtendedFinder method), 281
find() (robot.variables.finders.NumberFinder method), 281
Failed, 290
failed, 291
ExecutionFailed, 290
ExecutionFailures, 291
ExecutionPassed, 291
ExecutionResult (class in robot.libraries.Process), 72
ExecutionResult() (in module robot.result.resultbuilder), 230
ExecutionResultBuilder (class in robot.result.resultbuilder), 230
Exit (class in robot.running.status), 258
exit_for_loop() (robot.libraries.BuiltIn.BuiltIn method), 27
exit_for_loop_if() (robot.libraries.BuiltIn.BuiltIn method), 27
exit_on_error (robot.conf.settings.RobotSettings attribute), 16
exit_on_error_message (robot.running.status.TestMessage attribute), 259
exit_on_failure (robot.conf.settings.RobotSettings attribute), 16
exit_on_failure_message (robot.running.status.TestMessage attribute), 259
exit_on_fatal_message (robot.running.status.TestMessage attribute), 259
ExitForLoop, 292
expect() (robot.libraries.Telnet.TelnetConnection method), 89
extend() (robot.model.imports.Imports method), 158
extend() (robot.model.itemlist.ItemList method), 159
extend() (robot.model.keyword.Keywords method), 161
extend() (robot.model.message.Messages method), 162
extend() (robot.model.testcase.TestCases method), 171
extend() (robot.model.testsuite.TestSuites method), 173
ExtendedFinder (class in robot.variables.finders), 281
fail() (in module robot.utils.asserts), 265
fail() (robot.libraries.BuiltIn.BuiltIn method), 27
fail() (robot.output.filelogger.FileLogger method), 180
fail() (robot.output.logger.Logger method), 184
fail() (robot.output.loggerhelper.AbstractLogger method), 184
fail() (robot.output.output.Output method), 185
failed (robot.model.stats.Stat attribute), 166
Failure (class in robot.running.status), 258
failure_occurred() (robot.running.status.Exit method), 258
failures (robot.running.status.SuiteStatus attribute), 258
failures (robot.running.status.TestStatus attribute), 258
fatal_error() (robot.libraries.BuiltIn.BuiltIn method), 28
feed() (robot.libraries.Telnet.TelnetTerminalEmulator method), 91
feed() (robot.parsing.htmlreader.HtmlReader method), 191
fetch_from_left() (robot.libraries.String.StringReader method), 79
fetch_from_right() (robot.libraries.String.String method), 80
file() (robot.tidy.Tidy method), 299
file_should_be_empty() (robot.libraries.OperatingSystem.OperatingSystem method), 61
file_should_exist() (robot.libraries.OperatingSystem.OperatingSystem method), 60
file_should_not_be_empty() (robot.libraries.OperatingSystem.OperatingSystem method), 61
file_should_not_exist() (robot.libraries.OperatingSystem.OperatingSystem method), 60
FileWriter() (in module robot.writer.filewriters), 287
fill_named() (robot.running.arguments.argumentmapper.KeywordCallTemplate method), 240
fill_positional() (robot.running.arguments.argumentmapper.KeywordCallTemplate method), 240
fill_rawq() (robot.libraries.Telnet.TelnetConnection method), 89
Filter (class in robot.model.filter), 157
filter() (robot.model.testsuite.TestSuite method), 173
filter() (robot.output.pyloggingconf.RobotHandler method), 186
filter() (robot.result.model.TestSuite method), 229
filter() (robot.running.model.TestSuite method), 254
filter_messages() (robot.result.model.TestSuite method), 228
final_argument_whitespace (robot.parsing.restsupport.CaptureRobotData attribute), 198
find() (robot.variables.finders.EmptyFinder method), 281
find() (robot.variables.finders.EnvironmentFinder method), 281
find() (robot.variables.finders.ExtendedFinder method), 281
find() (robot.variables.finders.NumberFinder method), 281
find() (robot.variables.finders.StoredFinder method), 281
find() (robot.variables.finders.VariableFinder method), 281
find_all() (robot.libraries.XML.ElementFinder method), 102
find_coercers() (robot.running.arguments.javaargumentcoercer.CoercerFinder method), 242
find_file() (in module robot.utils.robotpath), 277
find_recommendations() (robot.utils.recommendations.RecommendationFinder method), 275
FirstColumnAligner (class in robot.writer.aligners), 286
Fixture (class in robot.parsing.settings), 199
flatten_keywords (robot.conf.settings.RobotSettings attribute), 17
flatten_keywords (robot.conf.settings.RobotSettings attribute), 17
FlattenByNameMatcher (class in robot.result.flattenkeywordmatcher), 215
FlattenByTagMatcher (class in robot.result.flattenkeywordmatcher), 215
FlattenByTypeMatcher (class in robot.result.flattenkeywordmatcher), 215
flavor (robot.running.model.ForLoop attribute), 249
FloatCoercer (class in robot.running.arguments.javaargumentcoercer), 242
flush() (robot.output.console.highlighting.HighlightingStream method), 178
flush() (robot.output.pyloggingconf.RobotHandler method), 186
focus() (robot.librariesdialogs_py.InputDialog method), 119
focus() (robot.librariesdialogs_py.MessageDialog method), 106
focus() (robot.librariesdialogs_py.PassFailDialog method), 145
focus() (robot.librariesdialogs_py.SelectionDialog method), 132
focus_get() (robot.librariesdialogs_py.InputDialog method), 119
focus_get() (robot.librariesdialogs_py.MessageDialog method), 106
focus_get() (robot.librariesdialogs_py.PassFailDialog method), 145
focus_get() (robot.librariesdialogs_py.SelectionDialog method), 132
focus_lastfor() (robot.librariesdialogs_py.MessageDialog method), 106
focus_lastfor() (robot.librariesdialogs_py.PassFailDialog method), 145
focus_lastfor() (robot.librariesdialogs_py.SelectionDialog method), 132
focus_set() (robot.librariesdialogs_py.InputDialog method), 119
focus_set() (robot.librariesdialogs_py.MessageDialog method), 106
focus_set() (robot.librariesdialogs_py.PassFailDialog method), 145
focus_set() (robot.librariesdialogs_py.SelectionDialog method), 132
focusmodel() (robot.librariesdialogs_py.InputDialog method), 119
focusmodel() (robot.librariesdialogs_py.MessageDialog method), 106
focusmodel() (robot.librariesdialogs_py.PassFailDialog method), 145
focusmodel() (robot.librariesdialogs_py.SelectionDialog method), 132
FOR_ITEM_TYPE (robot.model.keyword.Keyword attribute), 159
FOR_ITEM_TYPE (robot.result.model.Keyword attribute), 225
FOR_ITEM_TYPE (robot.running.model.ForLoop attribute), 249
FOR_LOOP_TYPE (robot.model.keyword.Keyword attribute), 159
FOR_LOOP_TYPE (robot.result.model.Keyword attribute), 225
FOR_LOOP_TYPE (robot.running.model.ForLoop attribute), 249
focus_displayof() (robot.librariesdialogs_py.InputDialog method), 119
focus_displayof() (robot.librariesdialogs_py.MessageDialog method), 106
focus_displayof() (robot.librariesdialogs_py.PassFailDialog method), 145
focus_displayof() (robot.librariesdialogs_py.SelectionDialog method), 132
focus_force() (robot.librariesdialogs_py.InputDialog method), 119
focus_force() (robot.librariesdialogs_py.MessageDialog method), 106
focus_force() (robot.librariesdialogs_py.PassFailDialog method), 145
focus_force() (robot.librariesdialogs_py.SelectionDialog method), 132
ForInEnumerateRunner (class in robot.running.steprunner), 260
ForInRangeRunner (class in robot.running.steprunner), 260
ForInRangeRunner (class in robot.running.steprunner), 260
ForInRunner (class in robot.running.steprunner), 260
ForInZipRunner (class in robot.running.steprunner), 260
ForLoop (class in robot.parsing.model), 196
ForLoop (class in robot.running.model), 249
ForLoopItemsRemover (class in robot.result.keywordremover), 219
ForLoopPopulator (class in robot.parsing.tablepopulators), 203
format() (robot.output.pyloggingconf.RobotHandler method), 186
format() (robot.tidy.ArgumentValidator method), 300
format() (robot.utils.htmlformatters.HeaderFormatter method), 270
format() (robot.utils.htmlformatters.LineFormatter method), 270
format() (robot.utils.htmlformatters.RulerFormatter method), 270
format() (robot.utils.htmlformatters.TableFormatter method), 271
format() (robot.utils.htmlformatters.PreformattedFormatter method), 271
format() (robot.utils.htmlformatters.ListFormatter method), 270
format() (robot.utils.htmlformatters.ParagraphFormatter method), 270
format() (robot.utils.htmlformatters.TableFormatter method), 271
format() (robot.utils.htmlformatters.HeaderFormatter method), 270
format() (robot.utils.htmlformatters.RulerFormatter method), 270
format() (robot.utils.htmlformatters.TableFormatter method), 271
format() (robot.utils.unic.PrettyRepr method), 279
format_assign_message() (in module robot.utils.text), 279
format_header() (robot.writer.formatters.PipeFormatter method), 288
format_header() (robot.writer.formatters.TsvFormatter method), 288
format_header() (robot.writer.formatters.TxtFormatter method), 288
format_header() (robot.writer.htmlformatter.HtmlFormatter method), 288
format_time() (in module robot.utils.robottime), 277
format_url() (robot.utils.htmlformatters.LinkFormatter method), 270
ForRunner() (in module robot.running.steprunner), 260
frame() (robot.libraries.dialogs_py.InputDialog method), 119
frame() (robot.libraries.dialogs_py.MessageDialog method), 106
frame() (robot.libraries.dialogs_py.PassFailDialog method), 145
frame() (robot.libraries.dialogs_py.SelectionDialog method), 132
FrameworkError, 289
frange() (in module robot.utils.frange), 270
FromDirectoryPopulator (class in robot.parsing.populators), 197
FromFilePopulator (class in robot.parsing.populators), 197
fromkeys() (robot.utils.dotdict.DotDict method), 267
fromkeys() (robot.utils.ordereddict.OrderedDict method), 275
full_message (robot.result.model.TestSuite attribute), 227
G
gather_failed_suites() (in module robot.conf.gatherfailed), 16
gather_failed_tests() (in module robot.conf.gatherfailed), 16
GatherFailedSuites (class in robot.conf.gatherfailed), 15
GatherFailedTests (class in robot.conf.gatherfailed), 15
generate_random_string() (robot.libraries.String.String method), 80
GeneratorWriter (class in robot.htmldata.htmlfilewriter), 18
go() (robot.libraries.dialogs_py.InputDialog method), 119
go() (robot.libraries.dialogs_py.MessageDialog method), 106
go() (robot.libraries.dialogs_py.PassFailDialog method), 145
go() (robot.libraries.dialogs_py.SelectionDialog method), 132
get() (robot.model.metadata.Metadata method), 162
get() (robot.utils.dotdict.DotDict method), 267
get() (robot.utils.normalizing.NormalizedDict method), 274
generate_random_string() (robot.libraries.String.String method), 80
get_arguments() (robot.output.listenerarguments.EndKeywordArguments method), 181
get_arguments() (robot.output.listenerarguments.EndSuiteArguments method), 181
get_result() (robot.running.timeouts.ironpython.Runner method), 284
get_runner() (robot.running.namespace.KeywordStore method), 256
get_runner() (robot.running.namespace.Namespace method), 255
get_selection_from_user() (in module robot.libraries.Dialogs), 57
get_setter() (robot.parsing.model.InitFileSettingTable method), 194
get_setter() (robot.parsing.model.ResourceFileSettingTable method), 193
get_setter() (robot.parsing.model.TestCase method), 195
get_setter() (robot.parsing.model.TestCaseFileSettingTable method), 193
get_setter() (robot.parsing.model.UserKeyword method), 196
get_slice_from_list() (robot.libraries.Collections.Collections method), 49
get_socket() (robot.libraries.Telnet.TelnetConnection method), 89
get_starttag_text() (robot.parsing.htmlreader.HtmlReader method), 191
get_stat() (robot.model.tagstatistics.TagStatInfo method), 170
get_substring() (robot.libraries.String.String method), 80
get_system_encoding() (in module robot.utils.encodingsniffer), 269
get_test_values() (robot.running.defaults.TestDefaults method), 245
get_time() (in module robot.utils.robottime), 277
get_time() (robot.libraries.BuiltIn.BuiltIn method), 28
get_timestamp() (in module robot.utils.robottime), 278
get_value_from_user() (in module robot.libraries.Dialogs), 57
get_variable_value() (robot.libraries.BuiltIn.BuiltIn method), 29
get_variables() (robot.libraries.BuiltIn.BuiltIn method), 29
get_version() (in module robot.version), 301
getboolean() (robot.libraries.dialogs_py.InputDialog method), 120
getboolean() (robot.libraries.dialogs_py.MessageDialog attribute), 106
getboolean() (robot.libraries.dialogs_py.PassFailDialog attribute), 145
getboolean() (robot.libraries.dialogs_py.SelectionDialog attribute), 132
getint (robot.libraries.dialogs_py.MessageDialog attribute), 106
getint (robot.libraries.dialogs_py.PassFailDialog attribute), 145
getint (robot.libraries.dialogs_py.SelectionDialog attribute), 132
getparser() (robot.libraries.Remote.TimeoutTransport method), 74
getpos() (robot.parsing.htmlreader.HtmlReader method), 191
getvar() (robot.libraries.dialogs_py.InputDialog method), 120
getvar() (robot.libraries.dialogs_py.MessageDialog method), 107
getvar() (robot.libraries.dialogs_py.PassFailDialog method), 146
getvar() (robot.libraries.dialogs_py.SelectionDialog method), 133
getKeywordArguments (class in robot.running.dynamicmethods), 245
getKeywordDocumentation (class in robot.running.dynamicmethods), 245
getKeywordNames (class in robot.running.dynamicmethods), 245
getKeywordTags (class in robot.running.dynamicmethods), 245
getpos() (robot.parsing.htmlreader.HtmlReader method), 191
getvar() (robot.libraries.dialogs_py.MessageDialog attribute), 106
getvar() (robot.libraries.dialogs_py.PassFailDialog attribute), 145
getvar() (robot.libraries.dialogs_py.SelectionDialog attribute), 132
getdoc() (in module robot.utils.misc), 274
getdouble (robot.libraries.dialogs_py.MessageDialog attribute), 106
getdouble (robot.libraries.dialogs_py.PassFailDialog attribute), 145
getdouble (robot.libraries.dialogs_py.SelectionDialog attribute), 132
GlobalScope (class in robot.running.libraryscopes), 247
getparser() (robot.libraries.Remote.TimeoutTransport method), 74
getpos() (robot.parsing.htmlreader.HtmlReader method), 191
getvar() (robot.libraries.dialogs_py.MessageDialog method), 107
getvar() (robot.libraries.dialogs_py.PassFailDialog method), 146
getvar() (robot.libraries.dialogs_py.SelectionDialog method), 133
GlobalVariables (class in robot.variables.scopes), 283
goahead() (robot.parsing.htmlreader.HtmlReader method), 191
grab_current() (robot.libraries.dialogs_py.InputDialog method), 120
grab_current() (robot.libraries.dialogs_py.MessageDialog method), 107
grab_current() (robot.libraries.dialogs_py.PassFailDialog method), 146
grab_current() (robot.libraries.dialogs_py.SelectionDialog method), 133
grab_release() (robot.libraries.dialogs_py.InputDialog method), 120
grab_release() (robot.libraries.dialogs_py.MessageDialog method), 107
grab_release() (robot.libraries.dialogs_py.PassFailDialog method), 146
grab_release() (robot.libraries.dialogs_py.SelectionDialog method), 133
330 Index
grab_set() (robot.libraries.dialogs_py.InputDialog method), 120
grab_set() (robot.libraries.dialogs_py.MessageDialog method), 107
grab_set() (robot.libraries.dialogs_py.PassFailDialog method), 146
grab_set() (robot.libraries.dialogs_py.SelectionDialog method), 133
grab_set_global() (robot.libraries.dialogs_py.InputDialog method), 120
grab_set_global() (robot.libraries.dialogs_py.MessageDialog method), 107
grab_set_global() (robot.libraries.dialogs_py.PassFailDialog method), 146
grab_set_global() (robot.libraries.dialogs_py.SelectionDialog method), 133
green() (robot.libraries.OperatingSystem.OperatingSystem method), 59
grip() (robot.libraries.dialogs_py.InputDialog method), 120
grip() (robot.libraries.dialogs_py.MessageDialog method), 107
grip() (robot.libraries.dialogs_py.PassFailDialog method), 146
grip() (robot.libraries.dialogs_py.SelectionDialog method), 133
grip_bbox() (robot.libraries.dialogs_py.InputDialog method), 120
grip_bbox() (robot.libraries.dialogs_py.MessageDialog method), 107
grip_bbox() (robot.libraries.dialogs_py.PassFailDialog method), 146
grip_bbox() (robot.libraries.dialogs_py.SelectionDialog method), 133
grip_columnconfigure() (robot.libraries.dialogs_py.SelectionDialog method), 133
grip_rowconfigure() (robot.libraries.dialogs_py.InputDialog method), 120
grip_rowconfigure() (robot.libraries.dialogs_py.MessageDialog method), 107
grip_rowconfigure() (robot.libraries.dialogs_py.PassFailDialog method), 146
grip_rowconfigure() (robot.libraries.dialogs_py.SelectionDialog method), 133
grip_slaves() (robot.libraries.dialogs_py.InputDialog method), 120
grip_slaves() (robot.libraries.dialogs_py.MessageDialog method), 107
grip_slaves() (robot.libraries.dialogs_py.PassFailDialog method), 146
grip_slaves() (robot.libraries.dialogs_py.SelectionDialog method), 133
grip_size() (robot.libraries.dialogs_py.InputDialog method), 120
grip_size() (robot.libraries.dialogs_py.MessageDialog method), 107
grip_size() (robot.libraries.dialogs_py.PassFailDialog method), 146
grip_size() (robot.libraries.dialogs_py.SelectionDialog method), 133

handle() (robot.output.pyloggingconf.RobotHandler method), 186
handle() (robot.running.arguments.argumentresolver.DictToKwargs method), 241
handle() (robot.running.arguments.javaargumentcoercer.VarargsHandler method), 242
handle_charref() (robot.parsing.htmlreader.HtmlReader method), 190
handle_comment() (robot.parsing.htmlreader.HtmlReader method), 191
handle_data() (robot.parsing.htmlreader.HtmlReader method), 190
handle_decl() (robot.parsing.htmlreader.HtmlReader method), 191
handle_endtag() (robot.parsing.htmlreader.HtmlReader method), 190
handle_entityref() (robot.parsing.htmlreader.HtmlReader method), 190
handle_imports() (robot.running.namespace.Namespace method), 255
handle_pi() (robot.parsing.htmlreader.HtmlReader method), 190
handle_startendtag() (robot.parsing.htmlreader.HtmlReader method), 191
handle_starttag() (robot.parsing.htmlreader.HtmlReader method), 190
handle_suite_teardown_failures() (robot.result.executionresult.CombinedResult method), 214
handle_suite_teardown_failures() (robot.result.executionresult.Result method), 214
handle_suite_teardown_failures() (robot.result.executionresult.TestSuite method), 229
handleError() (robot.output.pyloggingconf.RobotHandler method), 186
Handler() (in module robot.running.handlers), 246
HandlerExecutionFailed, 290
HandlerStore (class in robot.running.handlerstore), 246
handles() (robot.htmldata.htmlfilewriter.CssFileWriter method), 19
handles() (robot.htmldata.htmlfilewriter.GeneratorWriter method), 18
handles() (robot.htmldata.htmlfilewriter.JsFileWriter method), 19
handles() (robot.htmldata.htmlfilewriter.LineWriter method), 18
handles() (robot.htmldata.htmlfilewriter.ModelWriter method), 18
handles() (robot.htmldata.jsonwriter.DictDumper method), 19
handles() (robot.htmldata.jsonwriter.IntegerDumper method), 19
handles() (robot.htmldata.jsonwriter.MappingDumper method), 19
handles() (robot.htmldata.jsonwriter.NoneDumper method), 20
handles() (robot.htmldata.jsonwriter.StringDumper method), 19
handles() (robot.htmldata.jsonwriter.TupleListDumper method), 19
handles() (robot.libdocpkg.consoleviewer.ConsoleViewer class method), 20
handles() (robot.libdocpkg.htmlwriter.LibdocModelWriter method), 21
handles() (robot.reporting.logreportwriters.RobotModelWriter method), 206
handles() (robot.running.arguments.javaargumentcoercer.BooleanCoercer method), 242
handles() (robot.running.arguments.javaargumentcoercer.FloatCoercer method), 242
handles() (robot.running.arguments.javaargumentcoercer.IntegerCoercer method), 242
handles() (robot.running.arguments.javaargumentcoercer.NullCoercer method), 242
handles() (robot.testdoc.TestdocModelWriter method), 299
handles() (robot.utils.htmlformatters.HeaderFormatter method), 270
handles() (robot.utils.htmlformatters.LineFormatter method), 270
handles() (robot.utils.htmlformatters.ListFormatter method), 271
handles() (robot.utils.htmlformatters.ParagraphFormatter method), 271
handles() (robot.utils.htmlformatters.PreformattedFormatter method), 271
handles() (robot.utils.htmlformatters.RulerFormatter method), 270
handles() (robot.utils.htmlformatters.TableFormatter method), 271
handles() (robot.utils.importer.ByPathImporter method), 272
handles() (robot.utils.importer.DottedImporter method), 272
handles() (robot.utils.importer.NonDottedImporter method), 272
has_content (robot.parsing.restsupport.CaptureRobotData attribute), 198
has_data() (robot.parsing.model.Variable method), 195
has_data() (robot.parsing.restsupport.RobotDataStorage method), 198
has_key() (robot.utils.dotdict.DotDict method), 267
has_key() (robot.utils.ordereddict.OrderedDict method), 275
has_tests() (robot.parsing.model.TestCaseFile method), 192
has_tests() (robot.parsing.model.TestDataDirectory method), 193
head (robot.parsing.datarow.DataRow attribute), 189
header (robot.parsing.model.InitFileSettingTable attribute), 194
header (robot.parsing.model.KeywordTable attribute), 195
header (robot.parsing.model.ResourceFileSettingTable attribute), 194
header (robot.parsing.model.TestCaseFileSettingTable attribute), 193
header (robot.parsing.model.TestCaseTable attribute), 195
HeaderCell (class in robot.writer.htmlformatter), 288
HeaderFormatter (class in robot.utils.htmlformatters), 270
highlight() (robot.output.console.highlighting.HighlightingStream method), 178
Highlighter() (in module robot.output.console.highlighting), 178
HighlightingStream (class in robot.output.console.highlighting), 178
html (robot.model.message.Message attribute), 161
html (robot.output.loggerhelper.Message attribute), 185
html (robot.result.model.Message attribute), 224
html() (robot.libdocpkg.htmlwriter.DocFormatter method), 21
html() (robot.reporting.jsbuildingcontext.JsBuildingContext method), 204
html_column_count (robot.writer.datafilewriter.WritingContext attribute), 287
html_escape() (in module robot.utils.markuputils), 272
html_format (robot.writer.datafilewriter.WritingContext attribute), 287
html_format() (in module robot.utils.markuputils), 272
html_message (robot.model.message.Message attribute), 161
html_message (robot.output.loggerhelper.Message attribute), 185
html_message (robot.result.model.Message attribute), 224
HtmlCell (class in robot.writer.htmlformatter), 288
HtmlFileWriter (class in robot.htmldata.htmlfilewriter), 18
HtmlFileWriter (class in robot.htmldata.htmlfilewriters), 272
HtmlFormatter (class in robot.utils.htmlformatters), 270
HtmlFormatter (class in robot.writer.htmlformatter), 288
HtmlReader (class in robot.parsing.htmlreader), 190
HtmlTemplate (class in robot.htmldata.jartemplate), 19
HtmlTemplate (class in robot.htmldata.normaltemplate), 20
HtmlWriter (class in robot.utils.markupwriters), 272
iconbitmap() (robot.libraries.dialogs_py.MessageDialog method), 108
iconbitmap() (robot.libraries.dialogs_py.PassFailDialog method), 147
iconbitmap() (robot.libraries.dialogs_py.SelectionDialog method), 134
iconify() (robot.libraries.dialogs_py.InputDialog method), 121
iconify() (robot.libraries.dialogs_py.MessageDialog method), 108
iconify() (robot.libraries.dialogs_py.PassFailDialog method), 147
iconify() (robot.libraries.dialogs_py.SelectionDialog method), 134
iconmask() (robot.libraries.dialogs_py.InputDialog method), 121
iconmask() (robot.libraries.dialogs_py.MessageDialog method), 108
iconmask() (robot.libraries.dialogs_py.PassFailDialog method), 147
iconmask() (robot.libraries.dialogs_py.SelectionDialog method), 134
iconname() (robot.libraries.dialogs_py.InputDialog method), 121
iconname() (robot.libraries.dialogs_py.MessageDialog method), 108
iconname() (robot.libraries.dialogs_py.PassFailDialog method), 147
iconname() (robot.libraries.dialogs_py.SelectionDialog method), 134
iconposition() (robot.libraries.dialogs_py.InputDialog method), 121
iconposition() (robot.libraries.dialogs_py.MessageDialog method), 108
iconposition() (robot.libraries.dialogs_py.PassFailDialog method), 147
iconposition() (robot.libraries.dialogs_py.SelectionDialog method), 134
iconwindow() (robot.libraries.dialogs_py.InputDialog method), 121
iconwindow() (robot.libraries.dialogs_py.MessageDialog method), 108
iconwindow() (robot.libraries.dialogs_py.PassFailDialog method), 147
iconwindow() (robot.libraries.dialogs_py.SelectionDialog method), 134
id (robot.model.keyword.Keyword attribute), 160
id (robot.model.stats.SuiteStat attribute), 166
id (robot.model.testcase.TestCase attribute), 171
id (robot.model.testsuite.TestSuite attribute), 172
id (robot.result.model.Keyword attribute), 225
id (robot.result.model.TestCase attribute), 226
id (robot.result.model.TestSuite attribute), 229
id (robot.running.model.ForLoop attribute), 250

Index 333
Index 335

info() (robot.parsing.restsupport.CaptureRobotData
method), 198
info() (robot.utils.application.DefaultLogger
method), 262
Information, 290
InitFileSettingTable (class in robot.parsing.model), 194
InitHandler() (in module robot.running.handlers), 246
INITIAL (robot.parsing.htmlreader.HtmlReader
attribute), 190
inplace() (robot.tidy.Tidy method), 300
InputDialog (class in robot.libraries.dialogs_jy), 103
InputDialog (class in robot.libraries.dialogs_py), 116
insert() (robot.model.imports.Imports method), 158
insert() (robot.model.itemlist.ItemList method), 159
insert() (robot.model.keyword.Keywords method), 161
insert() (robot.model.message.Messages method), 162
insert() (robot.model.testcase.TestCases method), 171
insert() (robot.model.testsuite.TestSuites method), 173
insert_into_list() (robot.libraries.Collections.Collections
method), 49
IntegerCoercer (class in
robot.running.arguments.javaargumentcoercer), 242
IntegerDumper (class in robot.htmldata.jsonwriter), 19
interact() (robot.libraries.Telnet.TelnetConnection
method), 89
INTERNAL_UPDATE_FREQUENCY
(robot.libraries.Telnet.TelnetConnection
attribute), 86
invalidate_import_caches() (in module
robot.utils.importer), 271
InvalidForRunner (class in robot.running.steprunner), 260
is_active() (robot.parsing.settings.Fixture method), 199
is_active() (robot.parsing.settings.Library method), 201
is_for_loop() (robot.parsing.settings.Arguments method), 200
is_for_loop() (robot.parsing.settings.Documentation
method), 199
is_for_loop() (robot.parsing.settings.Fixture method), 199
is_for_loop() (robot.parsing.settings.Library method), 201
is_for_loop() (robot.parsing.settings.Metadata method), 201
is_for_loop() (robot.parsing.settings.Resource method), 201
is_for_loop() (robot.parsing.settings.Setting method), 198
is_for_loop() (robot.parsing.settings.Tags method), 200
is_for_loop() (robot.parsing.settings.Template method), 199
is_for_loop() (robot.parsing.settings.Timeout method), 200
is_for_loop() (robot.parsing.settings.Variables method), 202
is_global (robot.running.libraryscopes.GlobalScope
attribute), 247
is_global (robot.running.libraryscopes.TestCaseScope
attribute), 248
is_global (robot.running.libraryscopes.TestSuiteScope
attribute), 247
is_indented() (robot.parsing.datarow.DataRow method), 190
is_integer() (in module robot.libraries.datatypes), 278
is_java_init() (in module robot.libraries.robotinspect), 276
is_java_method() (in module robot.libraries.robotinspect),
276
is_list_like() (in module robot.libraries.datatypes), 278
is_list_var() (in module robot.variables.isvar), 282
is_list_variable() (robot.variables.splitter.VariableSplitter
method), 284
is_number() (in module robot.libraries.datatypes), 278
is_process_running() (robot.libraries.Process.Process
method), 70
is_scalar_var() (in module robot.variables.isvar), 282
is_set() (robot.parsing.model.ForLoop method), 196
is_set() (robot.parsing.model.Step method), 196
is_set() (robot.parsing.settings.Arguments method), 199
is_set() (robot.parsing.settings.Documentation method), 199
is_set() (robot.parsing.settings.Library method), 201
is_set() (robot.parsing.settings.Setting method), 195
is_set() (robot.parsing.settings.Variables method), 200
is_set() (robot.parsing.settings.Timeout method), 198
is_set() (robot.parsing.settings.Tags method), 200
is_set() (robot.parsing.settings.Template method), 199
JavaArgumentCoercer (class in robot.running.arguments.javaargumentcoercer), 242
JavaArgumentParser (class in robot.running.arguments.argumentparser), 240
JavaCapturer (class in robot.running.outputcapture), 256
JavaDocBuilder (class in robot.libdocpkg.javabuilder), 21
JavaDocBuilder() (in module robot.libdocpkg.builder), 20
JavaErrorDetails (class in robot.utils.error), 269
join_command_line() (robot.libraries.Process.Process method), 72
join_path() (robot.libraries.OperatingSystem.OperatingSystem method), 64
join_paths() (robot.libraries.OperatingSystem.OperatingSystem method), 64
join_string_with_value() (robot.parsing.settings.StringValueJoiner method), 198
JsBuildingContext (class in robot.reporting.jsbuildingcontext), 204
JsExecutionResult (class in robot.reporting.jsexecutionresult), 205
JsFileWriter (class in robot.htmldata.htmlfilewriter), 18
JsModelBuilder (class in robot.reporting.jsmodelbuilders), 205
JsonConverter (class in robot.libdocpkg.htmlwriter), 21
JsonConverter (class in robot.testdoc), 299
JsonDumper (class in robot.htmldata.jsonwriter), 19
JsResultWriter (class in robot.reporting.jsresultwriter), 206
keep_in_dictionary() (robot.libraries.Collections.Collections method), 49
keys() (robot.libraries.dialogs_py.InputDialog method), 121
keys() (robot.libraries.dialogs_py.MessageDialog method), 108
keys() (robot.libraries.dialogs_py.PassFailDialog method), 147
keys() (robot.libraries.dialogs_py.SelectionDialog method), 134
keys() (robot.model.metadata.Metadata method), 162
keys() (robot.utils.dotdict.DotDict method), 268
keys() (robot.utils.normalizing.NormalizedDict method), 274
keys() (robot.utils.ordereddict.OrderedDict method), 275
keys() (robot.parsing.settings.Fixture attribute), 199
keywords() (robot.parsing.settings.Fixture attribute), 199
keywords() (robot.utils.dotdict.DotDict method), 268
keywords() (robot.utils.normalizing.NormalizedDict method), 274
keywords() (robot.utils.ordereddict.OrderedDict method), 275
Keyword (class in robot.model.keyword), 159
Keyword (class in robot.result.model), 224
Keyword (class in robot.running.keyword), 224
Keyword (class in robot.running.model), 248
Keyword (class in robot.running.process), 248
Keyword (robot.parsing.settings.Fixture attribute), 199
Keyword() (in module robot.api.deco), 12
Keyword_class (robot.model.keyword.Keyword attribute), 159
metadata (robot.model.testsuite.TestSuite attribute), 172
metadata (robot.result.model.TestSuite attribute), 229
metadata (robot.running.model.TestSuite attribute), 254
MetadataHandler (class in robot.result.xmlelementhandlers), 237
MetadataItemHandler (class in robot.result.xmlelementhandlers), 237
MetadataList (class in robot.parsing.settings), 202
MetadataPopulator (class in robot.parsing.tablepopulators), 204
minargs (robot.running.arguments.argumentspec.ArgumentSpec attribute), 241
minsize() (robot.libraries.dialogs_pyInputDialog method), 121
minsize() (robot.libraries.dialogs_py.MessageDialog method), 108
minsize() (robot.libraries.dialogs_py.PassFailDialog method), 147
minsize() (robot.libraries.dialogs_py.SelectionDialog method), 134
mode_and_arguments() (robot.tidy.ArgumentValidator method), 300
ModelCombiner (class in robot.running.runner), 258
ModelModifier (class in robot.model.modifier), 163
ModelObject (class in robot.model.modelobject), 163
ModelWriter (class in robot.htmldata.htmlfilewriter), 18
move_directory() (robot.libraries.OperatingSystem.OperatingSystem method), 63
move_files() (robot.libraries.OperatingSystem.OperatingSystem method), 63
mro() (robot.utils.setter.SetterAwareType method), 279
msg() (robot.libraries.Telnet.TelnetConnection method), 89
mt_interact() (robot.libraries.Telnet.TelnetConnection method), 90
MultiMatcher (class in robot.utils.match), 273

name (robot.model.keyword.Keyword attribute), 160
name (robot.model.stats.Stat attribute), 165
name (robot.model.testcase.TestCase attribute), 170
name (robot.model.testsuite.TestSuite attribute), 172
name (robot.output.pyloggingconf.RobotHandler attribute), 186
name (robot.parsing.model.InitFileSettingTable attribute), 194
name (robot.parsing.model.KeywordTable attribute), 195
name (robot.parsing.model.ResourceFile attribute), 192
name (robot.parsing.model.ResourceFileSettingTable attribute), 194
name (robot.parsing.model_TestCaseFile attribute), 192
name (robot.parsing.model_TestCaseFileSettingTable attribute), 193
name (robot.parsing.model_TestCaseTable attribute), 195
name (robot.parsing.model_TestCaseDirectory attribute), 193
name (robot.parsing.model_VariableTable attribute), 194
name (robot.result.model.Keyword attribute), 224
name (robot.result.model_TestCase attribute), 227
name (robot.result.model_TestCaseSuite attribute), 229
name (robot.running.dynamicmethods.GetKeywordArguments attribute), 245
name (robot.running.dynamicmethods.GetKeywordDocumentation attribute), 245
name (robot.running.dynamicmethods.GetKeywordNames attribute), 245
name (robot.running.dynamicmethods.GetKeywordTags attribute), 245
name (robot.running.dynamicmethods.RunKeyword attribute), 245
name (robot.running.model_TestCaseLoop attribute), 250
name (robot.running.model_TestCaseSuite attribute), 249
name (robot.running.model_TestCaseSuite attribute), 251
NameCell (class in robot.writer.htmlformatter), 288
Namespace (class in robot.running.namespace), 255
NamespaceStripper (class in robot.libraries.XML), 102
nametowidget() (robot.libraries.dialogs_pyInputDialog method), 121
nametowidget() (robot.libraries.dialogs_py.MessageDialog method), 108
nametowidget() (robot.libraries.dialogs_py.PassFailDialog method), 147
nametowidget() (robot.libraries.dialogs_py.SelectionDialog method), 134
NEW_ENVIRON_IS (robot.libraries.Telnet.TelnetConnection attribute), 86
NEW_ENVIRON_VALUE (robot.libraries.Telnet.TelnetConnection attribute), 86
NEW_ENVIRON_VAR (robot.libraries.Telnet.TelnetConnection attribute), 86
new_suite_scope() (robot.output.listenermethods.LibraryListenerMethods method), 182
new_suite_scope() (robot.output.listenermethods.LibraryListenerMethods method), 182
newline (robot.utils.htmlformatters.LineFormatter attribute), 270
no_dynamic_method() (in module robot.running.dynamicmethods), 245
no_operation() (robot.libraries.BuiltIn.BuiltIn method), 31
NoConnection (class in robot.utils.connectioncache), 267
NoHighlighting (class in robot.output.console.highlighting), 178
NoMatchError, 91
non_ascii (robot.libraries.Remote.ArgumentCoercer attribute), 73
non_critical (robot.model.stats.TagStat attribute), 167
non_critical (robot.model.tagstatistics.TagStatistics attribute), 169
non_critical_tags (robot.conf.settings.RebotSettings attribute), 18
non_critical_tags (robot.conf.settings.RobotSettings attribute), 17
NonDottedImporter (class in robot.utils.importer), 272
none_shall_pass() (in module robot.libraries.Easter), 57
NoOutput (class in robot.output.console.quiet), 179
NoReturnValueResolver (class in robot.variables.assigner), 280
normal (robot.model.keyword.Keywords attribute), 161
normalize() (in module robot.utils.normalizing), 274
normalize() (robot.parsing.model.InitFileSettingTable method), 194
normalize() (robot.parsing.model.ResourceFileSettingTable method), 194
normalize() (robot.parsing.model.TestCase method), 195
normalize() (robot.parsing.model.TestCaseFileSettingTable method), 193
normalize() (robot.parsing.model.UserKeyword method), 196
normalize_path() (robot.libraries.OperatingSystem.OperatingSystem method), 64
NormalizedDict (class in robot.utils.normalizing), 274
normpath() (in module robot.utils.robotpath), 276
NotSet (class in robot.libraries.Collections), 45
NotTagPattern (class in robot.model.tags), 168
NullAligner (class in robot.writer.aligners), 286
NullCoercer (class in robot.running.arguments.javaargumentcoercer), 242
NullMarkupWriter (class in robot.libraries.markupwriters), 273
NullNamedArgumentResolver (class in robot.running.arguments.argumentresolver), 241
NullPopulator (class in robot.parsing.tablepopulators), 202
NumberFinder (class in robot.variables.finders), 281
numerator (robot.reporting.stringcache.StringIndex attribute), 208

O
OldStyleSettingAndVariableTableHeaderMatcher (class in robot.parsing.model), 197
OldStyleTestAndKeywordTableHeaderMatcher (class in robot.parsing.model), 197
OneReturnValueResolver (class in robot.variables.assigner), 280
open() (robot.libraries.Telnet.TelnetConnection method), 90
open_connection() (robot.libraries.Telnet.Telnet method), 85
OperatingSystem (class in robot.libraries.OperatingSystem), 57
option_add() (robot.libraries.dialogs_py.InputDialog method), 121
option_add() (robot.libraries.dialogs_py.MessageDialog method), 108
option_add() (robot.libraries.dialogs_py.PassFailDialog method), 147
option_add() (robot.libraries.dialogs_py.SelectionDialog method), 134
option_clear() (robot.libraries.dialogs_py.InputDialog method), 121
option_clear() (robot.libraries.dialogs_py.MessageDialog method), 108
option_clear() (robot.libraries.dialogs_py.PassFailDialog method), 147
option_clear() (robot.libraries.dialogs_py.SelectionDialog method), 134
option_get() (robot.libraries.dialogs_py.InputDialog method), 122
option_get() (robot.libraries.dialogs_py.MessageDialog method), 109
option_get() (robot.libraries.dialogs_py.PassFailDialog method), 148
option_get() (robot.libraries.dialogs_py.SelectionDialog method), 135
option_readfile() (robot.libraries.dialogs_py.InputDialog method), 122
option_readfile() (robot.libraries.dialogs_py.MessageDialog method), 109
option_readfile() (robot.libraries.dialogs_py.PassFailDialog method), 148
option_readfile() (robot.libraries.dialogs_py.SelectionDialog method), 135
option_spec (robot.parsing.restsupport.CaptureRobotData attribute), 198
optional_arguments (robot.parsing.restsupport.CaptureRobotData attribute), 198
OrderedDict (class in robot.utils.orderecdict), 275
OrTagPattern (class in robot.model.tags), 168
Output (class in robot.output.output), 185
output (robot.conf.settings.RebotSettings attribute), 18
output (robot.conf.settings.RobotSettings attribute), 17
output() (robot.output.console.verbose.VerboseWriter method), 179
output_directory (robot.conf.settings.RebotSettings attribute), 18
output_directory (robot.conf.settings.RobotSettings attribute), 17
output_file() (robot.output.console.dotted.DottedOutput method), 177
output_file() (robot.output.console.verbose.VerboseOutput method), 179
output_file() (robot.output.filelogger.FileLogger method), 180
output_file() (robot.output.listeners.LibraryListeners method), 182
output_file() (robot.output.listeners.Listeners method), 182
output_file() (robot.output.listeners.LibraryListeners method), 183
OutputCapturer (class in robot.running.outputcapture), 256
OutputWriter (class in robot.running.outputwriter), 206
overrideredirect() (robot.libraries.dialogs_py.InputDialog method), 122
overrideredirect() (robot.libraries.dialogs_py.MessageDialog method), 109
overrideredirect() (robot.libraries.dialogs_py.PassFailDialog method), 148
overrideredirect() (robot.libraries.dialogs_py.SelectionDialog method), 135

pack_propagate() (robot.libraries.dialogs_py.InputDialog method), 122
pack_propagate() (robot.libraries.dialogs_py.MessageDialog method), 109
pack_propagate() (robot.libraries.dialogs_py.PassFailDialog method), 148
pack_propagate() (robot.libraries.dialogs_py.SelectionDialog method), 135
pack_slaves() (robot.libraries.dialogs_py.InputDialog method), 122
pack_slaves() (robot.libraries.dialogs_py.MessageDialog method), 109
pack_slaves() (robot.libraries.dialogs_py.PassFailDialog method), 148
pack_slaves() (robot.libraries.dialogs_py.SelectionDialog method), 135
pad_console_length() (in module robot.utils.text), 279
ParagraphFormatter (class in robot.utils.htmlformatters), 270
parent (robot.model.keyword.Keyword attribute), 160
parent (robot.model.message.Message attribute), 161
parent (robot.model.testcase.TestCase attribute), 170
parent (robot.model.testsuite.TestSuite attribute), 172
parent (robot.output.loggerhelper.Message attribute), 185
parent (robot.result.model.Keyword attribute), 225
parent (robot.result.model.Message attribute), 224

parent (robot.result.model.TestCase attribute), 227
parent (robot.result.model.TestSuite attribute), 229
parent (robot.running.model.ForLoop attribute), 250
parent (robot.running.model.Keyword attribute), 249
parent (robot.running.model_TestCase attribute), 251
parent (robot.running.model_TestCase attribute), 254
ParentMessage (class in robot.running.status), 259
parse() (robot.running.arguments.argumentparser.DynamicArgumentParser method), 240
parse() (robot.running.arguments.argumentparser.JavaArgumentParser method), 240
parse() (robot.running.arguments.argumentparser.PythonArgumentParser method), 240
parse() (robot.running.arguments.argumentparser.UserKeywordArgumentParser method), 240
parse() (robot.running.arguments.embedded.EmbeddedArgumentParser method), 242
parse_args() (robot.utils.argumentparser.ArgumentParser method), 263
parse_arguments() (robot.libdoc.LibDoc method), 294
parse_arguments() (robot.rebot.Rebot method), 295
parse_arguments() (robot.run.RobotFramework method), 296
parse_arguments() (robot.testdoc.TestDoc method), 298
parse_arguments() (robot.tidy.TidyCommandLine method), 300
parse_arguments() (robot.utils.application.Application method), 262
parse_bogus_comment() (robot.parsing.htmlreader.HtmlReader method), 191
parse_comment() (robot.parsing.htmlreader.HtmlReader method), 191
Parse_declaration() (robot.parsing.htmlreader.HtmlReader method), 191
parse_endtag() (robot.parsing.htmlreader.HtmlReader method), 191
parse_html_declaration() (robot.parsing.htmlreader.HtmlReader method), 191
parse_marked_section() (robot.parsing.htmlreader.HtmlReader method), 191
parse_image() (robot.parsing.htmlreader.HtmlReader method), 191
parse_p() (robot.parsing.htmlreader.HtmlReader method), 191
parse_p() (robot.parsing.htmlreader.HtmlReader method), 191
parse_response() (robot.libraries.Remote.TimeoutTransport method), 74
parse_starttag() (robot.parsing.htmlreader.HtmlReader method), 191
parse_time() (in module robot.utils.robottime), 277
parse_xml() (robot.libraries.XML.XML method), 95
pass_execution() (robot.libraries.BuiltIn.BuiltIn method), 31
pass_execution_if() (robot.libraries.BuiltIn.BuiltIn method), 32
passed (robot.model.stats.Stat attribute), 166
passed (robot.result.model.Keyword attribute), 225
passed (robot.result.model.TestCase attribute), 226
passed (robot.result.model.TestSuite attribute), 227
PassedKeywordRemover (class in robot.result.keywordremover), 216
PassExecution, 291
PassFailDialog (class in robot.libraries.dialogs_jy), 103
PassFailDialog (class in robot.libraries.dialogs_py), 142
path_to_url() (in module robot.utils.robotpath), 276
pause_execution() (in module robot.libraries.Dialogs), 56
pformat() (robot.utils.unic.PrettyRepr method), 279
PipeFormatter (class in robot.writer.formatters), 288
PipeSeparatedTxtWriter (class in robot.writer.filewriters), 287
place_slaves() (robot.libraries.dialogs_py.InputDialog method), 122
place_slaves() (robot.libraries.dialogs_py.MessageDialog method), 109
place_slaves() (robot.libraries.dialogs_py.PassFailDialog method), 148
place_slaves() (robot.libraries.dialogs_py.SelectionDialog method), 135
plural_or_not() (in module robot.utils.misc), 274
pop() (robot.model.imports.Imports method), 158
pop() (robot.model.itemlist.ItemList method), 159
pop() (robot.model.keyword.Keywords method), 161
pop() (robot.model.message.Messages method), 162
pop() (robot.model.metadata.Metadata method), 162
pop() (robot.model.testcase.TestCases method), 171
pop() (robot.model.testsuite.TestSuites method), 174
pop() (robot.utils.dotdict.DotDict method), 268
pop() (robot.utils.normalizing.NormalizedDict method), 275
pop() (robot.utils.ordereddict.OrderedDict method), 275
pop_from_dictionary() (robot.libraries.Collections.Collection method), 50
popitem() (robot.model.metadata.Metadata method), 162
popitem() (robot.utils.dotdict.DotDict method), 268
popitem() (robot.utils.normalizing.NormalizedDict method), 275
popitem() (robot.utils.ordereddict.OrderedDict method), 275
populate() (robot.parsing.model.ResourceFile method), 192
populate() (robot.parsing.model.TestCaseFile method), 192
populate() (robot.parsing.model.TestDataDirectory method), 193
populate() (robot.parsing.populators.FromDirectoryPopulator method), 197
populate() (robot.parsing.populators.FromFilePopulator method), 197
populate() (robot.parsing.populators.UserKeywordPopulator method), 200
populate() (robot.parsing.settings.Arguments method), 200
populate() (robot.parsing.settings.Documentation method), 199
populate() (robot.parsing.settings.Fixture method), 199
populate() (robot.parsing.settings.Library method), 201
populate() (robot.parsing.settings.Metadata method), 201
populate() (robot.parsing.settings-metadata.MetadataList method), 202
populate() (robot.parsing.settings.Resource method), 201
populate() (robot.parsing.settings.Return method), 201
populate() (robot.parsing.settings.Setting method), 198
populate() (robot.parsing.settings.Tags method), 200
populate() (robot.parsing.settings.Template method), 199
populate() (robot.parsing.settings.Timeout method), 200
populate() (robot.parsing.settings.Variables method), 202
populate() (robot.parsing.tablepopulators.DocumentationPopulator method), 203
populate() (robot.parsing.tablepopulators.ForLoopPopulator method), 203
populate() (robot.parsing tableaupopulators.KeywordTablePopulator method), 203
populate() (robot.parsing tableaupopulators.MetaDataPopulator method), 204
populate() (robot.parsing tableaupopulators.NullPopulator method), 202
populate() (robot.parsing tableaupopulators.Populator method), 202
populate() (robot.parsing tableaupopulators.SettingPopulator method), 203
populate() (robot.parsing tableaupopulators.SettingTablePopulator method), 203
populate() (robot.parsing tableaupopulators.StepPopulator method), 204
populate() (robot.parsing tableaupopulators.TestCasePopulator method), 203
populate() (robot.parsing tableaupopulators.TestTablePopulator method), 203
populate() (robot.parsing tableaupopulators.UserKeywordPopulator method), 203
populate() (robot.parsing tableaupopulators.VariablePopulator method), 203
populate() (robot.parsing tableaupopulators.VariableTablePopulator method), 203
populate_library() (robot.parsing.settings.ImportList method), 202
populate_resource() (robot.parsing.settings.ImportList method), 202
populate_variables() (robot.parsing.settings.ImportList method), 202
positionfrom() (robot.libraries.dialogs_py.InputDialog method), 122
positionfrom() (robot.libraries.dialogs_py.MessageDialog method), 109
method), 109
positionfrom() (robot.libraries.dialogs_py.PassFailDialog
method), 148
positionfrom() (robot.libraries.dialogs_py.SelectionDialog
method), 135
pprint() (robot.utils.unic.PrettyRepr method), 280
pre_rebot_modifiers (robot.conf.settings.RebotSettings
attribute), 18
pre_rebot_modifiers (robot.conf.settings.RobotSettings
attribute), 17
pre_run_modifiers (robot.conf.settings.RobotSettings
attribute), 17
PreformattedFormatter (class in
robot.utils.htmlformatters), 271
prepr() (in module robot.utils.unic), 279
PrettyRepr (class in robot.utils.unic), 279
printable_name() (in module robot.utils.misc), 274
Process (class in robot.libraries.Process), 67
PROCESS (robot.parsing.htmlreader.HtmlReader at-
tribute), 190
process() (robot.utils.argumentparser.ArgFileParser
method), 263
process_empty_suite (robot.conf.settings.RebotSettings
attribute), 18
process_rawq() (robot.libraries.Telnet.TelnetConnection
method), 90
process_should_be_running() (robot.libraries.Process.Process
method), 70
process_should_be_stopped() (robot.libraries.Process.Process
method), 70
ProcessConfiguration (class in robot.libraries.Process), 73
propagate() (robot.libraries.dialogs_py.InputDialog
method), 122
propagate() (robot.libraries.dialogs_py.MessageDialog
method), 109
propagate() (robot.libraries.dialogs_py.PassFailDialog
method), 148
propagate() (robot.libraries.dialogs_py.SelectionDialog
method), 135
protocol() (robot.libraries.dialogs_py.InputDialog
method), 122
protocol() (robot.libraries.dialogs_py.MessageDialog
method), 109
protocol() (robot.libraries.dialogs_py.PassFailDialog
method), 148
protocol() (robot.libraries.dialogs_py.SelectionDialog
method), 135
prune_input() (robot.reporting.jsbuildingcontext.JsBuildingContext
method), 205
py2to3() (in module robot.utils.compat), 266
PythonArgumentParser (class in
robot.running.arguments.argumentparser), 240
PythonCapturer (class in robot.running.outputcapture), 256
PythonErrorDetails (class in robot.utils.error), 269
PythonImporter (class in robot.variables.filesetter), 281
Q
QuietOutput (class in robot.output.console.quiet), 179
quit() (robot.libraries.dialogs_py.InputDialog method), 122
quit() (robot.libraries.dialogs_py.MessageDialog
method), 109
quit() (robot.libraries.dialogs_py.PassFailDialog
method), 148
quit() (robot.libraries.dialogs_py.SelectionDialog
method), 135
R
raise_error() (robot.utils.connectioncache.NoConnection
method), 267
randomize() (robot.running.model.TestSuite method), 252
randomize_seed (robot.conf.settings.RobotSettings
attribute), 16
randomize_suites (robot.conf.settings.RobotSettings
attribute), 16
randomize_tests (robot.conf.settings.RobotSettings
attribute), 16
Randomizer (class in robot.running.randomizer), 256
rawq_getchar() (robot.libraries.Telnet.TelnetConnection
method), 90
read() (robot.libraries.Telnet.TelnetConnection method), 88
read() (robot.libraries.Telnet.TerminalEmulator method), 91
read() (robot.parsing.htmlreader.HtmlReader method), 190
read() (robot.parsing.tsvreader.TsvReader method), 204
read() (robot.parsing.txtreader.TxtReader method), 204
read() (robot.utils.utf8reader.Utf8Reader method), 280
read() (robot.variables.tablesetter.VariableTableReader
method), 284
read_all() (robot.libraries.Telnet.TelnetConnection
method), 90
read_eager() (robot.libraries.Telnet.TelnetConnection
method), 90
read_lazy() (robot.libraries.Telnet.TelnetConnection
method), 90
read_sb_data() (robot.libraries.Telnet.TelnetConnection
method), 90
read_some() (robot.libraries.Telnet.TelnetConnection
method), 90
read_until() (robot.libraries.Telnet.TelnetConnection
method), 88
read_until() (robot.libraries.Telnet.TerminalEmulator method), 91
read_until_prompt() (robot.libraries.Telnet.TelnetConnection method), 89
read_until_regexp() (robot.libraries.Telnet.TelnetConnection method), 88
read_until_regexp() (robot.libraries.Telnet.TerminalEmulator method), 91
read_very_eager() (robot.libraries.Telnet.TelnetConnection method), 90
read_very_lazy() (robot.libraries.Telnet.TelnetConnection method), 90
readlines() (robot.utils.utf8reader.Utf8Reader method), 280
real (robot.reporting.stringcache.StringIndex attribute), 209
Rebot (class in robot.rebot), 295
rebot() (in module robot), 11
rebot() (in module robot.rebot), 296
rebot_cli() (in module robot), 11
rebot_cli() (in module robot.rebot), 295
RebotSettings (class in robot.conf.settings), 17
recommend_similar_keywords() (robot.running.namespace.KeywordRecommendationFinder method), 256
RecommendationFinder (class in robot.utils.recommendations), 275
red() (robot.output.console.highlighting.AnsiHighlighter method), 178
red() (robot.output.console.highlighting.DosHighlighter method), 178
red() (robot.output.console.highlighting.NoHighlighting method), 178
regexp_escape() (robot.libraries.BuiltIn.BuiltIn method), 32
register() (robot.librariesdialogs_py.InputDialog method), 122
register() (robot.librariesdialogs_py.MessageDialog method), 109
register() (robot.librariesdialogs_py.PassFailDialog method), 148
register() (robot.librariesdialogs_py.SelectionDialog method), 135
register() (robot.output.listenermethods.LibraryListenerMethods method), 182
register() (robot.output.listeners.LibraryListeners method), 182
register() (robot.utils.connectioncache.ConnectionCache method), 267
register_console_logger() (robot.output.logger.Logger method), 185
register_listeners() (robot.output.logger.Logger method), 183
register_logger() (robot.output.logger.Logger method), 183
register_run_keyword() (in module robot.libraries.BuiltIn), 44
register_syslog() (robot.output.logger.Logger method), 183
register_xml_logger() (robot.output.logger.Logger method), 183
relative_source() (robot.reporting.jsbuildingcontext.JsBuildingContext method), 205
release() (robot.output.pyloggingconf.RobotHandler method), 186
release() (robot.running.outputcapture.JavaCapturer method), 256
release() (robot.running.outputcapture.PythonCapturer method), 256
reload_library() (robot.libraries.BuiltIn.BuiltIn method), 32
reload_library() (robot.running.namespace.Namespace method), 255
Remote (class in robot.libraries.Remote), 73
RemoteError, 293
RemoteResult (class in robot.libraries.Remote), 73
RemovalMessage (class in robot.result.keywordremover), 221
remove() (robot.variables.store.VariableStore method), 284
remove_data_not_needed_in_report() (robot.reporting.jsexecutionresult.JsExecutionResult method), 205
remove_directory() (robot.libraries.OperatingSystem.OperatingSystem method), 62
remove_duplicates() (robot.libraries.Collections.Collections method), 50
remove_element() (robot.libraries.XML.XML method), 101
remove_element_attribute() (robot.libraries.XML.XML method), 100
remove_element_attributes() (robot.libraries.XML.XML method), 100
remove_elements() (robot.libraries.XML.XML method), 101
remove_elements_attribute() (robot.libraries.XML.XML method), 100
remove_elements_attributes() (robot.libraries.XML.XML method), 100
remove_empty_suites() (robot.model.testsuite.TestSuite method), 173
remove_empty_suites() (robot.result.model.TestSuite method), 229
remove_empty_suites() (robot.running.model.TestSuite method), 254
remove_environment_variable()
(robot.libraries.OperatingSystem.OperatingSystem method), 64
remove_file() (robot.libraries.OperatingSystem.OperatingSystem method), 62
remove_files() (robot.libraries.OperatingSystem.OperatingSystem method), 62
remove_from_dictionary()
(robot.libraries.Collections.Collections method), 50
remove_from_list() (robot.libraries.Collections.Collections method), 50
remove_keywords (robot.conf.settings.RebotSettings attribute), 18
remove_keywords (robot.conf.settings.RobotSettings attribute), 17
remove_keywords() (robot.result.model.TestSuite method), 228
remove_path() (in module robot.pythonpathsetter), 295
remove_string() (robot.libraries.String.String method), 79
remove_string_using_regexp()
(robot.libraries.String.String method), 79
remove_tags (robot.model.configurer.SuiteConfigurer attribute), 155
remove_tags() (robot.result.configurer.SuiteConfigurer attribute), 212
remove_tags() (robot.libraries.BuiltIn.BuiltIn method), 32
remove_values_from_list()
(robot.libraries.Collections.Collections method), 50
removeFilter() (robot.output.pyloggingconf.RobotHandler method), 187
RemoveKeywords (class in robot.result.resultbuilder), 230
repeat_keyword() (robot.libraries.BuiltIn.BuiltIn method), 32
replace() (robot.running.arguments.argumentresolver.VariableReplacer method), 241
replace_defaults() (robot.running.arguments.argumentmapper.KeywordCallTemplate method), 240
replace_list() (robot.variables.replacer.VariableReplacer method), 282
replace_list() (robot.variables.scopes.GlobalVariables method), 283
replace_list() (robot.variables.scopes.VariableScopes method), 283
replace_list() (robot.variables.variables.Variables method), 285
replace_scalar() (robot.variables.scopes.VariableScopes method), 283
replace_scalar() (robot.variables.variables.Variables method), 285
report (robot.conf.settings.RebotSettings attribute), 18
report (robot.conf.settings.RobotSettings attribute), 17
report() (robot.output.console.dotted.StatusReporter method), 177
report_invalid_syntax() (robot.model.imports.Import method), 158
report_invalid_syntax() (robot.parsing.model.InitFileSettingTable method), 194
report_invalid_syntax() (robot.parsing.model.KeywordTable method), 194
report_invalid_syntax() (robot.parsing.model.ResourceFile method), 192
report_invalid_syntax() (robot.parsing.model.ResourceFileSettingTable method), 194
report_invalid_syntax() (robot.parsing.model.TestCase method), 195
report_invalid_syntax() (robot.parsing.model.TestCaseFile method), 193
report_invalid_syntax() (robot.parsing.model.TestCaseFileSettingTable method), 195
report_invalid_syntax() (robot.parsing.model.VariableTableValueBase method), 284
report_invalid_syntax() (robot.parsing.model.VariableTableValueBase method), 285
Index 349
robot.utils.importer (module), 271
robot.utils.markuputils (module), 272
robot.utils.markupwriters (module), 272
robot.utils.match (module), 273
robot.utils.misce (module), 273
robot.utils.normalizing (module), 274
robot.utils.ordereddict (module), 275
robot.utils.platform (module), 275
robot.utils.recommendations (module), 275
robot.utils.robotenv (module), 276
robot.utils.robotinspect (module), 276
robot.utils.robotio (module), 276
robot.utils.robotpath (module), 276
robot.utils.robottime (module), 277
robot.utils.robottypes (module), 278
robot.utils.robottypes2 (module), 278
robot.utils.setter (module), 279
robot.utils.sortable (module), 279
robot.utils.text (module), 279
robot.utils.unic (module), 279
robot.utils.utf8reader (module), 280
robot.variables (module), 280
robot.variables.assigner (module), 280
robot.variables.filesetter (module), 281
robot.variables.finders (module), 281
robot.variables.isvar (module), 282
robot.variables.notfound (module), 282
robot.variables.replacer (module), 282
robot.variables.scopes (module), 282
robot.variables.splitter (module), 284
robot.variables.store (module), 284
robot.variables.tablesetter (module), 284
robot.variables.variables (module), 285
robot.version (module), 301
robot.writer (module), 285
robot.writer.aligners (module), 286
robot.writer.dataextractor (module), 286
robot.writer.datafilewriter (module), 286
robot.writer.filewriters (module), 287
robot.writer.formatters (module), 288
robot.writer.htmlformatter (module), 288
robot.writer.htmltemplate (module), 289
robot.writer.rowsplitter (module), 289
robot_format (robot.writer.datafilewriter.WritingContext attribute), 287
robot_handler_enabled() (in module robot.output.pyloggingconf), 186
ROBOT_LIBRARY_SCOPE (robot.libraries.BuiltIn.BuiltIn attribute), 58
ROBOT_LIBRARY_SCOPE (robot.libraries.Remote.Remote attribute), 73
ROBOT_LIBRARY_SCOPE (robot.libraries.Reserved.Reserved attribute), 74
ROBOT_LIBRARY_SCOPE (robot.libraries.Screenshot.Screenshot attribute), 75
ROBOT_LIBRARY_SCOPE (robot.libraries.String.String attribute), 76
ROBOT_LIBRARY_SCOPE (robot.libraries.Telnet.Telnet attribute), 85
ROBOT_LIBRARY_SCOPE (robot.libraries.XML.XML attribute), 95
ROBOT_LIBRARY_VERSION (robot.libraries.BuiltIn.BuiltIn attribute), 24
ROBOT_LIBRARY_VERSION (robot.libraries.Collections.Collections attribute), 46
ROBOT_LIBRARY_VERSION (robot.libraries.OperatingSystem.OperatingSystem attribute), 58
ROBOT_LIBRARY_VERSION (robot.libraries.Remote.Remote attribute), 73
ROBOT_LIBRARY_VERSION (robot.libraries.Reserved.Reserved attribute), 74
ROBOT_LIBRARY_VERSION (robot.libraries.Screenshot.Screenshot attribute), 75
ROBOT_LIBRARY_VERSION (robot.libraries.String.String attribute), 76
ROBOT_LIBRARY_VERSION (robot.libraries.Telnet.Telnet attribute), 85
ROBOT_LIBRARY_VERSION (robot.libraries.XML.XML attribute), 95
ROBOT_LIBRARY_VERSION (robot.libraries.OperatingSystem.OperatingSystem attribute), 58
ROBOT_LIBRARY_VERSION (robot.libraries.Screenshot.Screenshot attribute), 75
ROBOT_LIBRARY_VERSION (robot.libraries.String.String attribute), 76
ROBOT_LIBRARY_VERSION (robot.libraries.Telnet.Telnet attribute), 85
ROBOT_LIBRARY_VERSION (robot.libraries.XML.XML attribute), 95
ROBOT_LIBRARY_VERSION (robot.libraries.OperatingSystem.OperatingSystem attribute), 58
ROBOT_LIBRARY_VERSION (robot.libraries.Screenshot.Screenshot attribute), 75
ROBOT_LIBRARY_VERSION (robot.libraries.String.String attribute), 76
ROBOT_LIBRARY_VERSION (robot.libraries.Telnet.Telnet attribute), 85
ROBOT_LIBRARY_VERSION (robot.libraries.XML.XML attribute), 95
robotDataStorage (class in robot.parsing.restitutionsupport), 198
RobotError, 289
RobotFramework (class in robot.run), 296
RobotHandler (class in robot.output.pyloggingconf), 186
RobotHandler (class in robot.result.xmlelementhandlers), 235
RobotModelWriter (class in robot.reporting.logreportwriters), 206
RobotNotRunningError, 44

Index
set_newline() (robot.libraries.Telnet.TelnetConnection method), 86
set_option_negotiation_callback() (robot.libraries.Telnet.TelnetConnection method), 91
set_prompt() (robot.libraries.Telnet.TelnetConnection method), 86
set_screenshot_directory() (robot.libraries.Screenshot.Screenshot method), 75
set_search_order() (robot.running.namespace.Namespace method), 255
set_suite() (robot.variables.scopes.SetVariables method), 283
set_suite() (robot.variables.scopes.VariableScopes method), 283
set_suite_documentation() (robot.libraries.BuiltIn.BuiltIn method), 37
set_suite_metadata() (robot.libraries.BuiltIn.BuiltIn method), 37
set_suite_variable() (robot.libraries.BuiltIn.BuiltIn method), 37
set_tags() (robot.libraries.BuiltIn.BuiltIn method), 38
set_tags() (robot.model.testsuite.TestSuite method), 172
set_tags() (robot.result.model.TestSuite method), 229
set_tags() (robot.running.model.TestSuite method), 254
set_telnetlib_log_level() (robot.libraries.Telnet.TelnetConnection method), 87
set_test() (robot.variables.scopes.SetVariables method), 283
set_test() (robot.variables.scopes.VariableScopes method), 283
set_test_documentation() (robot.libraries.BuiltIn.BuiltIn method), 37
set_test_message() (robot.libraries.BuiltIn.BuiltIn method), 38
set_test_variable() (robot.libraries.BuiltIn.BuiltIn method), 38
set_timeout() (robot.libraries.Telnet.TelnetConnection method), 86
set_to_dictionary() (robot.libraries.Collections.Collections method), 51
set_variable() (robot.libraries.BuiltIn.BuiltIn method), 39
set_variable_if() (robot.libraries.BuiltIn.BuiltIn method), 39
setdefault() (robot.model.metadata.Metadata method), 163
setdefault() (robot.utils.dotdict.DotDict method), 268
setdefault() (robot.utils.normalizing.NormalizedDict method), 275
setdefault() (robot.utils.ordereddict.OrderedDict method), 275
setFormatter() (robot.output.pyloggingconf.RobotHandler method), 187
setLevel() (robot.output.pyloggingconf.RobotHandler method), 187
setter (class in robot.utils.setter), 279
SetterAwareType (class in robot.utils.setter), 279
Setting (class in robot.parsing.settings), 198
setting_name (robot.parsing.settings.Metadata attribute), 201
SettingPopulator (class in robot.parsing.tablepopulators), 203
settings (robot.parsing.model_TestCase attribute), 195
settings (robot.parsing.model.UserKeyword attribute), 196
SettingTablePopulator (class in robot.parsing.tablepopulators), 202
setup (robot.model.keyword.Keywords attribute), 161
setup_executed() (robot.running.status.SuiteStatus method), 258
setup_executed() (robot.running.status.TestStatus method), 259
setup_message (robot.running.status.ParentMessage attribute), 259
setup_message (robot.running.status.SuiteMessage attribute), 259
setup_message (robot.running.status.TestMessage attribute), 259
SETUP_TYPE (robot.model.keyword.Keyword attribute), 159
SETUP_TYPE (robot.result.model.Keyword attribute), 225
SETUP_TYPE (robot.running.model.ForLoop attribute), 250
SETUP_TYPE (robot.running.model.Keyword attribute), 248
setvar() (robot.libraries.dialogs_py.InputDialog method), 123
setvar() (robot.libraries.dialogs_py.MessageDialog method), 110
setvar() (robot.libraries.dialogs_py.PassFailDialog method), 149
setvar() (robot.libraries.dialogs_py.SelectionDialog method), 136
SetVariables (class in robot.variables.scopes), 283
severe() (robot.parsing.restsupport.CaptureRobotData method), 198
shortdoc (robot.libdocpkg.model.KeywordDoc attribute), 22
shortdoc (robot.running.usererrorhandler.UserErrorHandler attribute), 260
shortdoc (robot.running.userkeyword.EmbeddedArgumentsHandler attribute), 261
shortdoc (robot.running.userkeyword.UserKeywordHandler attribute), 261
should_be_byte_string() (robot.libraries.String.String method), 261
start() (robot.result.xmlelementhandlers.RootSuiteHandler method), 235
start() (robot.result.xmlelementhandlers.StatisticsHandler method), 238
start() (robot.result.xmlelementhandlers.SuiteHandler method), 235
start() (robot.result.xmlelementhandlers.SuiteStatusHandler method), 236
start() (robot.result.xmlelementhandlers.TagHandler method), 237
start() (robot.result.xmlelementhandlers.TagsHandler method), 237
start() (robot.result.xmlelementhandlers.TestCaseHandler method), 235
start() (robot.result.xmlelementhandlers.TestStatusHandler method), 236
start() (robot.result.xmlelementhandlers.TimeoutHandler method), 237
start() (robot.result.xmlelementhandlers.XmlElementHandler method), 235
start() (robot.running.timeouts.KeywordTimeout method), 243
start() (robot.running.timeouts.TimeoutHandler method), 243
start() (robot.utils.markupwriters.HtmlWriter method), 272
start() (robot.utils.markupwriters.NullMarkupWriter method), 273
start() (robot.utils.markupwriters.XmlWriter method), 273
start_errors() (robot.output.xmllogger.XmlLogger method), 187
start_keyword() (robot.conf.gatherfailed.GatherFailedSuites method), 16
start_keyword() (robot.conf.gatherfailed.GatherFailedTests method), 15
start_keyword() (robot.model.configurer.SuiteConfigurer method), 156
start_keyword() (robot.model.filter.EmptySuiteRemover method), 157
start_keyword() (robot.model.filter.Filter method), 158
start_keyword() (robot.model.modifier.ModelModifier method), 163
start_keyword() (robot.model.statistics.StatisticsBuilder method), 165
start_keyword() (robot.model.tagsetter.TagSetter method), 169
start_keyword() (robot.model.totalstatistics.TotalStatisticsBuilder method), 174
start_keyword() (robot.model.visitor.SuiteVisitor method), 176
start_keyword() (robot.output.console.dotted.StatusReporter method), 177
start_keyword() (robot.output.console.verbose.VerboseOutput method), 179
start_keyword() (robot.output.filelogger.FileLogger method), 180
start_keyword() (robot.output.logger.Logger method), 183
start_keyword() (robot.output.output.Output method), 185
start_keyword() (robot.output.xmllogger.XmlLogger method), 187
start_keyword() (robot.reporting.outputwriter.OutputWriter method), 207
start_keyword() (robot.reporting.xunitwriter.XUnitFileWriter method), 209
start_keyword() (robot.result.configurer.SuiteConfigurer method), 212
start_keyword() (robot.result.keywordremover.AllKeywordsRemover method), 215
start_keyword() (robot.result.keywordremover.ByTagKeywordRemover method), 218
start_keyword() (robot.result.keywordremover.DeletedKeywordRemover method), 219
start_keyword() (robot.result.keywordremover.WaitUntilKeywordSucceedsRemover method), 219
start_keyword() (robot.result.messagefilter.MessageFilter method), 222
start_keyword() (robot.result.resultbuilder.RemoveKeywords method), 231
start_keyword() (robot.result.suitetearndownfailed.SuiteTearndownFailed method), 232
start_keyword() (robot.result.suitetearndownfailed.SuiteTearndownFailureHandler method), 232
start_keyword() (robot.result.visitor.ResultVisitor method), 234
start_keyword() (robot.running.randomizer.Randomizer method), 256
start_keyword() (robot.running.runner.Runner method), 257
start_keyword() (robot.variables.scopes.SetVariables method), 283
start_keyword() (robot.variables.scopes.VariableScopes method), 358
start_suite() (robot.model.visitor.SuiteVisitor method), 176
start_suite() (robot.output.console.dotted.StatusReporter method), 177
start_suite() (robot.output.console.verbose.VerboseOutput method), 179
start_suite() (robot.output.filelogger.FileLogger method), 180
start_suite() (robot.output.logger.Logger method), 183
start_suite() (robot.output.output.Output method), 185
start_suite() (robot.output.xmllogger.XmlLogger method), 187
start_suite() (robot.reporting.outputwriter.OutputWriter method), 207
start_suite() (robot.result.configurer.SuiteConfigurer method), 212
start_suite() (robot.result.keywordremover.AllKeywordsRemover method), 216
start_suite() (robot.result.keywordremover.ByNameKeywordRemover method), 217
start_suite() (robot.result.keywordremover.ByTagKeywordRemover method), 218
start_suite() (robot.result.keywordremover.ForLoopItemsRemover method), 219
start_suite() (robot.result.keywordremover.PassedKeywordRemover method), 220
start_suite() (robot.result.keywordremover.WaitUntilKeywordSucceedsRemover method), 221
start_suite() (robot.result.keywordremover.WarningAndErrorFinder method), 222
start_suite() (robot.result.resultbuilder.RemoveKeywords method), 223
start_suite() (robot.result.resultbuilder.RemoveKeywords method), 230
start_suite() (robot.result.teardownfailed.SuiteTeardownFailed method), 232
start_suite() (robot.result.visitor.ResultVisitor method), 234
start_suite() (robot.running.context.ExecutionContexts method), 245
start_suite() (robot.running.libraryscopes.GlobalScope method), 247
start_suite() (robot.running.libraryscopes.TestCaseScope method), 248
start_suite() (robot.running.libraryscopes.TestSuiteScope method), 249
start_suite() (robot.running.namespace.Namespace method), 255
start_suite() (robot.running.randomizer.Randomizer method), 256
start_suite() (robot.running.runner.Runner method), 257
start_suite() (robot.variables.scopes.SetVariables method), 283
start_suite() (robot.variables.scopes.VariableScopes method), 285
start_suite_statistics() (robot.output.xmllogger.XmlLogger method), 187
start_suite_statistics() (robot.reporting.outputwriter.OutputWriter method), 207
start_suite_statistics() (robot.reporting.xunitwriter.XUnitFileWriter method), 209
start_test() (robot.conf.gatherfailed.GatherFailedSuites method), 16
start_test() (robot.conf.gatherfailed.GatherFailedTests method), 15
start_test() (robot.model.configurer.SuiteConfigurer method), 156
start_test() (robot.model.filter.EmptySuiteRemover method), 157
start_test() (robot.model.filter.EmptySuiteRemover method), 158
start_test() (robot.model.filter.EmptySuiteRemover method), 159
start_test() (robot.model.statistics.StatisticsBuilder method), 165
start_test() (robot.model.tagsetter.TagSetter method), 169
start_test() (robot.model.totalstatistics.TotalStatisticsBuilder method), 174
start_test() (robot.result.visitor.SuiteVisitor method), 177
start_test() (robot.output.console.dotted.StatusReporter method), 179
start_test() (robot.output.console.verbose.VerboseOutput method), 179
start_test() (robot.output.filelogger.FileLogger method), 180
start_test() (robot.output.logger.Logger method), 183
start_test() (robot.output.output.Output method), 185
start_test() (robot.output.xmllogger.XmlLogger method), 187
start_test() (robot.reporting.outputwriter.OutputStream method), 207
start_test() (robot.reporting.xunitwriter.XUnitWriter method), 210
start_test() (robot.result.configurer.SuiteConfigurer method), 212
start_test() (robot.result.keywordremover.AllKeywordsRemover method), 216
start_test() (robot.result.keywordremover.ByNameKeywordRemover method), 217
start_test() (robot.result.keywordremover.ByTagKeywordRemover method), 218
start_test() (robot.result.keywordremover.ForLoopItemsRemover method), 219
start_test() (robot.result.keywordremover.PassedKeywordRemover method), 220
start_test() (robot.result.keywordremover.WaitUntilKeywordRemover method), 221
start_test() (robot.result.keywordremover.WaitUntilKeywordRemover.method), 221
start_test() (robot.result.keywordremover.WaitUntilKeywordRemover method), 221
start_test() (robot.result.keywordremover.WaitUntilKeywordRemover method), 221
start_test() (robot.result.keywordremover.WaitUntilKeywordRemover method), 221
starts_for_loop() (robot.parsing.datarow.DataRow method), 190
StartKeywordArguments (class in robot.output.listenerarguments), 181
StartSuiteArguments (class in robot.output.listenerarguments), 181
StartTestArguments (class in robot.output.listenerarguments), 181
starttime (robot.result.model.Keyword attribute), 224
starttime (robot.result.model.TestCase attribute), 226
starttime (robot.result.model.TestSuite attribute), 227
stat (robot.model.suitestatistics.SuiteStatistics attribute), 168
stat (robot.model.suitestatistics.SuiteStatistics attribute), 168
state() (robot.libraries.dialogs_py.InputDialog method), 123
state() (robot.libraries.dialogs_py.MessageDialog method), 110
state() (robot.libraries.dialogs_py.PassFailDialog method), 149
state() (robot.libraries.dialogs_py.SelectionDialog method), 136
Statistics (class in robot.model.statistics), 164
Statistics (class in robot.model.statistics), 164
statistics (robot.result.executionresult.CombinedResult attribute), 214
statistics (robot.result.executionresult.Result attribute), 213
statistics (robot.result.model.TestSuite attribute), 227
statistics (robot.result.model.TestSuite attribute), 227
statistics (robot.result.model.TestSuite attribute), 227
statistics (robot.conf.settings.RebotSettings attribute), 18
statistics (robot.conf.settings.RobotSettings attribute), 17
StatisticsBuilder (class in robot.model.statistics), 165
StatisticsBuilder (class in robot.reporting.jsmodelbuilders), 205
StatisticsHandler (class in robot.reporting.jsmodelhandlers), 238
status (robot.errors.ContinueForLoop attribute), 292
status (robot.errors.ExecutionFailed attribute), 290
status (robot.errors.ExecutionFailures attribute), 291
status (robot.errors.ExecutionPassed attribute), 291
status (robot.errors.ExitForLoop attribute), 292
status (robot.errors.HandlerExecutionFailed attribute), 291
status (robot.errors.PassExecution attribute), 292
status (robot.errors.ReturnFromKeyword attribute), 293
status (robot.errors.UserKeywordExecutionFailed attribute), 291
status (robot.result.model.Keyword attribute), 224
status (robot.result.model.TestCase attribute), 226
status (robot.result.model.TestSuite attribute), 227
status (robot.running.status.SuiteStatus attribute), 258
status (robot.running.status.TestStatus attribute), 259
status() (robot.output.console verbose.VerboseWriter method), 179
status_rc (robot.conf.settings.RebotSettings attribute), 18
status_rc (robot.conf.settings.RobotSettings attribute), 17
StatusReporter (class in robot.output.console.dotted), 177
StatusReporter (class in robot.running.statusreporter), 259
stderr (robot.libraries.Process.ExecutionResult attribute), 73
stdout (robot.libraries.Process.ExecutionResult attribute), 73
StdoutLogSplitter (class in robot.output.stdoutlogsplitter), 187
Step (class in robot.parsing.model), 196
StepBuilder (class in robot.running.builder), 244
StepPopulator (class in robot.parsing.tablepopulators), 204
StepRunner (class in robot.running.steprunner), 259
StoredFinder (class in robot.variables.finders), 281
String (class in robot.libraries.String), 76
string() (robot.reporting.jsbuildingcontext.JsBuildingContext method), 204
string_value() (robot.parsing.settings.StringValueJoiner method), 198
StringCache (class in robot.reporting.stringcache), 209
StringDumper (class in robot.htmldata.jsonwriter), 19
StringIndex (class in robot.reporting.stringcache), 208
strings (robot.reporting.jsbuildingcontext.JsBuildingContext attribute), 205
StringValueJoiner (class in robot.parsing.settings), 198
strip() (robot.libraries.XML.NameSpaceStripper method), 102
strip_string() (robot.libraries.String.String method), 80
subtract_date_from_date() (in module robot.libraries.DateTime), 55
subtract_time_from_date() (in module robot.libraries.DateTime), 55
subtract_time_from_time() (in module robot.libraries.DateTime), 56
suite (robot.model.statistics.Statistics attribute), 164
suite (robot.result.executionresult.Result attribute), 213
suite_config (robot.conf.settings.RobotSettings attribute), 17
suite_separator() (robot.output.consoleverbose.VerboseWriter method), 179
suite_teardown_failed() (robot.result.model.TestSuite method), 230
SuiteBuilder (class in robot.reporting.jsmodelbuilders), 205
SuiteConfigurer (class in robot.model.configurer), 155
SuiteConfigurer (class in robot.result.configurer), 211
SuiteHandler (class in robot.result.xmlelementhandlers), 235
SuiteMessage (class in robot.running.status), 259
SuiteNamePatterns (class in robot.model.namepatterns), 164
suites (robot.model.suitestatistics.SuiteStatistics attribute), 168
suites (robot.model.suitestatistics.SuiteStatistics attribute), 168
suites (robot.result.model.suiteStatistics.attribute), 230
suites (robot.result.model.TestSuite attribute), 230
suites (robot.running.model.TestSuite attribute), 254
SuiteStat (class in robot.model.stats), 166
SuiteStatistics (class in robot.model.suitestatistics), 168
SuiteStatisticsBuilder (class in robot.model.suitestatistics), 168
SuiteStatus (class in robot.running.status), 258
SuiteStatusHandler (class in robot.result.xmlelementhandlers), 236
SuiteTeardownFailed (class in robot.result.suite teardownfailed), 232
SuiteTeardownFailureHandler (class in robot.result.suite teardownfailed), 231
SuiteVisitor (class in robot.model.visitor), 175
SuiteWriter (class in robot.reporting.jswriter), 206
supports_kwargs (robot.running.dynamicmethods.RunKeyword attribute), 245
switch() (robot.utils.connectioncache.ConnectionCache method), 267
switch_connection() (robot.libraries.Telnet.Telnet method), 85
system_decode() (in module robot.utils.encoding), 268
system_encode() (in module robot.utils.encoding), 268

T

table_end() (robot.parsing.htmlreader.HtmlReader method), 190
table_start() (robot.parsing.htmlreader.HtmlReader method), 190
TableFormatter (class in robot.utils.htmlformatters), 271
tag (robot.result.xmlelementhandlers.ArgumentHandler attribute), 238
tag (robot.result.xmlelementhandlers.ArgumentsHandler attribute), 238
tag (robot.result.xmlelementhandlers.AssignHandler attribute), 237
tag (robot.result.xmlelementhandlers.AssignVarHandler attribute), 238
tag (robot.result.xmlelementhandlers.DocHandler attribute), 236
tag (robot.result.xmlelementhandlers.ErrorsHandler attribute), 238
tag (robot.result.xmlelementhandlers.KeywordHandler attribute), 236
tag (robot.result.xmlelementhandlers.KeywordStatusHandler attribute), 236
tag (robot.result.xmlelementhandlers.MessageHandler attribute), 236
tag (robot.result.xmlelementhandlers.MetadataHandler attribute), 237
tag (robot.result.xmlelementhandlers.MetadataItemHandler attribute), 237
tag (robot.result.xmlelementhandlers.RobotHandler attribute), 235
tag (robot.result.xmlelementhandlers.RootSuiteHandler attribute), 235
tag (robot.result.xmlelementhandlers.SuiteHandler attribute), 235
tag (robot.result.xmlelementhandlers.SuiteStatusHandler attribute), 236
tag (robot.result.xmlelementhandlers.TagHandler attribute), 237
tag (robot.result.xmlelementhandlers.TagsHandler attribute), 237
tag (robot.result.xmlelementhandlers.TagSetter attribute), 169
tag (robot.result.xmlelementhandlers_TestCaseHandler attribute), 227
tag (robot.running.model.ForLoop attribute), 250
tag (robot.running.model.Keyword attribute), 249
tag (robot.running.model.TestCase attribute), 252
tag (robot.running.model.UserKeyword attribute), 255
tags (robot.result.model.Keyword attribute), 225
tags (robot.result.model_TestCase attribute), 227
tags (robot.running.model_TestCase attribute), 227
tags (robot.running.model_TestCase TestSuite attribute), 250
tags (robot.running.model_TestCase TestSuite attribute), 249
tags (robot.running.model_TestCase TestSuite attribute), 252
tags (robot.running.model_TestCase TestSuite attribute), 255
tags (robot.result.xmlelementhandlers_TestCase attribute), 237
tags (robot.result.xmlelementhandlers_TestCase attribute), 237
tags (robot.running.model_TestCase TestSuite attribute), 227
tags (robot.running.model_TestCase TestSuite attribute), 227
tags (robot.running.model_TestCase TestSuite attribute), 227
TagHandler (class in robot.result.xmlelementhandlers_TestCase), 237
TagStat (class in robot.result.stats_TestCase), 166
TagStatDoc (class in robot.result.tagstatistics_TestCase), 170
TagStatistics (class in robot.result.tagstatistics_TestCase), 169
TagStatisticsBuilder (class in robot.result.tagstatistics_TestCase), 170
TagStatLink (class in robot.result.tagstatistics_TestCase), 170
take_screenshot() (robot.libraries.Screenshot_TestCase.method), 75
take_screenshot_without_embedding() (robot.libraries.Screenshot_TestCase.method), 76
td_end() (robot.parsing.htmlreader_TestCase.HtmlReader method), 190
td_start() (robot.parsing.htmlreader_TestCase.HtmlReader method), 190
tearDown (robot.model_TestCase.Keywords attribute), 161
tearDown_allowed (robot.running_TestCase.Exit attribute), 258
tearDown_allowed (robot.running_TestCase_TestCase attribute), 258
tearDown_allowed (robot.running_TestCase_TestCase attribute), 259
tearDown_executed() (robot.running_TestCase_TestCase attribute), 258
tearDown_executed() (robot.running_TestCase_TestCase attribute), 259
tearDown_message (robot.running_TestCase_TestCase attribute), 259
tearDown_message (robot.running_TestCase_TestCase attribute), 259
TEAR_DOWN_TYPE (robot.model_TestCase.Keywords attribute), 159
TEAR_DOWN_TYPE (robot.result_TestCase.Keyword attribute), 225
TEAR_DOWN_TYPE (robot.running_TestCase_TestCase attribute), 250
TEAR_DOWN_TYPE (robot.running_TestCase_TestCase attribute), 248
telnet (class in robot.libraries_TestCase.Telnet), 81
telnetConnection (class in robot.libraries_TestCase.Telnet), 85
template (class in robot.parsing_TestCase), 199
template (robot.running_TestCase_TestCase attribute), 251
timeout (robot.errors.UserKeywordExecutionFailed attribute), 291
timeout (robot.model.keyword.Keyword attribute), 160
timeout (robot.model.testcase.TestCase attribute), 170
timeout (robot.result.model.Keyword attribute), 225
timeout (robot.result.model.TestCase attribute), 170
timeout (robot.running.model.ForLoop attribute), 250
timeout (robot.running.model.Keyword attribute), 249
timeout (robot.running.model.TestCase attribute), 251
timeout (robot.running.model.UserKeyword attribute), 255
TimeoutError, 290
TimeoutHandler (class in robot.result.xmlelementhandlers), 237
TimeoutTransport (class in robot.libraries.Remote), 74
timestamp (robot.model.message.Message attribute), 161
timestamp (robot.output.loggerhelper.Message attribute), 184
timestamp (robot.result.model.Message attribute), 224
timestamp() (robot.reporting.jsbuildingcontext.JsBuildingContext method), 205
timestamp_to_secs() (in module robot.utils.robottime), 278
TimestampCache (class in robot.utils.robottime), 278
timestr_to_secs() (in module robot.utils.robottime), 277
title() (robot.libraries.dialogs_pyInputDialog method), 123
title() (robot.libraries.dialogs_py.MessageDialog method), 110
title() (robot.libraries.dialogs_py.PassFailDialogue method), 149
title() (robot.libraries.dialogs_py.SelectionDialogue method), 136
tk_bisque() (robot.libraries.dialogs_pyInputDialog method), 123
tk_bisque() (robot.libraries.dialogs_py.MessageDialog method), 110
tk_bisque() (robot.libraries.dialogs_py.PassFailDialogue method), 149
tk_bisque() (robot.libraries.dialogs_py.SelectionDialogue method), 136
tk_focusFollowsMouse() (robot.libraries.dialogs_pyInputDialog method), 123
tk_focusFollowsMouse() (robot.libraries.dialogs_py.MessageDialog method), 110
tk_focusFollowsMouse() (robot.libraries.dialogs_py.PassFailDialogue method), 149
tk_focusFollowsMouse() (robot.libraries.dialogs_py.SelectionDialogue method), 136
tk_focusNext() (robot.libraries.dialogs_py.MessageDialog method), 110
tk_focusNext() (robot.libraries.dialogs_py.PassFailDialogue method), 149
tk_focusNext() (robot.libraries.dialogs_py.SelectionDialogue method), 137
tk_focusPrev() (robot.libraries.dialogs_pyInputDialog method), 124
tk_focusPrev() (robot.libraries.dialogs_py.MessageDialog method), 111
tk_focusPrev() (robot.libraries.dialogs_py.PassFailDialogue method), 150
tk_focusPrev() (robot.libraries.dialogs_py.SelectionDialogue method), 137
tk_menuBar() (robot.libraries.dialogs_pyInputDialog method), 124
tk_menuBar() (robot.libraries.dialogs_py.MessageDialog method), 111
tk_menuBar() (robot.libraries.dialogs_py.PassFailDialogue method), 150
tk_menuBar() (robot.libraries.dialogs_py.SelectionDialogue method), 137
tk_setPalette() (robot.libraries.dialogs_pyInputDialog method), 124
tk_setPalette() (robot.libraries.dialogs_py.MessageDialog method), 111
tk_setPalette() (robot.libraries.dialogs_py.PassFailDialogue method), 150
tk_setPalette() (robot.libraries.dialogs_py.SelectionDialogue method), 137
tk_strictMotif() (robot.libraries.dialogs_pyInputDialog method), 124
tk_strictMotif() (robot.libraries.dialogs_py.MessageDialog method), 111
tk_strictMotif() (robot.libraries.dialogs_py.PassFailDialogue method), 150
tk_strictMotif() (robot.libraries.dialogs_py.SelectionDialogue method), 137
tk_raise() (robot.libraries.dialogs_pyInputDialog method), 124
tk_raise() (robot.libraries.dialogs_py.MessageDialog method), 111
tk_raise() (robot.libraries.dialogs_py.PassFailDialogue method), 150
tk_raise() (robot.libraries.dialogs_py.SelectionDialogue method), 137
top (robot.running.context.ExecutionContexts attribute), 245
total (robot.model.statistics.Statistics attribute), 164
total (robot.model.stats.CombinedTagStat attribute), 167
total (robot.model.stats.CriticalTagStat attribute), 167
total (robot.model.stats.Stat attribute), 166
total (robot.model.stats.SuiteStat attribute), 166
total (robot.model.stats.TagStat attribute), 167
total (robot.model.stats.TotalStat attribute), 166
TotalStat (class in robot.model.stats), 166
TotalStatistics (class in robot.model.totalstatistics), 174
TotalStatisticsBuilder (class in robot.model.totalstatistics), 174
touch() (robot.libraries.OperatingSystem.OperatingSystem method), 66
tr_end() (robot.parsing.htmlreader.HtmlReader method), 190
tr_start() (robot.parsing.htmlreader.HtmlReader method), 190
trace() (in module robot.api.logger), 14
trace() (in module robot.output.librarylogger), 180
trace() (robot.output.filelogger.FileLogger method), 180
trace() (robot.output.logger.Logger method), 184
trace() (robot.output.loggerhelper.AbstractLogger method), 184
trace() (robot.output.output.Output method), 185
traceback (robot.utils.error.JavaErrorDetails attribute), 269
traceback (robot.utils.error.PythonErrorDetails attribute), 269
transient() (robot.libraries.dialogs_py.InputDialog method), 124
transient() (robot.libraries.dialogs_py.MessageDialog method), 111
transient() (robot.libraries.dialogs_py.PassFailDialog method), 150
transient() (robot.libraries.dialogs_py.SelectionDialog method), 137
tsv_column_count (robot.writer.datafilewriter.WritingContext attribute), 287
tsv_format (robot.writer.datafilewriter.WritingContext attribute), 287
TsvFileWriter (class in robot.writer.filewriters), 287
TsvFormatter (class in robot.writer.formatters), 287
TsvReader (class in robot.parsing.tsvreader), 204
TupleListDumper (class in robot.htmldata.jsonwriter), 19
txt_column_count (robot.writer.datafilewriter.WritingContext attribute), 287
txt_format (robot.writer.datafilewriter.WritingContext attribute), 287
TxtFormatter (class in robot.writer.formatters), 288
TxtReader (class in robot.parsing.txtreader), 204
type (robot.model.keyword.Keyword attribute), 160
type (robot.model.stats.CombinedTagStat attribute), 167
type (robot.model.stats.CriticalTagStat attribute), 167
type (robot.model.stats.SuiteStat attribute), 166
type (robot.model.stats.TagStat attribute), 166
type (robot.model.stats.TotalStat attribute), 166
type (robot.parsing.model.InitFileSettingTable attribute), 194
type (robot.parsing.model.KeywordTable attribute), 195
type (robot.parsing.model.ResourceFileSettingTable attribute), 194
type (robot.parsing.model.TestCaseFileSettingTable attribute), 193
type (robot.parsing.model.TestCaseTable attribute), 194
type (robot.parsing.model.VariableTable attribute), 194
type (robot.parsing.settings.Library attribute), 201
type (robot.parsing.settings.Resource attribute), 202
type (robot.parsing.settings.Variables attribute), 202
type (robot.result.model.Keyword attribute), 225
type (robot.running.model.ForLoop attribute), 251
type (robot.running.model.Keyword attribute), 249
type (robot.running.timeouts.KeywordTimeout attribute), 243
type (robot.running.timeouts.TestTimeout attribute), 242
type_name() (in module robot.utils.robottypes2), 278
unbind() (robot.libraries.dialogs_py.InputDialog method), 124
unbind() (robot.libraries.dialogs_py.MessageDialog method), 111
unbind() (robot.libraries.dialogs_py.PassFailDialog method), 150
unbind() (robot.libraries.dialogs_py.SelectionDialog method), 137
unbind_all() (robot.libraries.dialogs_py.InputDialog method), 124
unbind_all() (robot.libraries.dialogs_py.MessageDialog method), 111
unbind_all() (robot.libraries.dialogs_py.PassFailDialog method), 150
unbind_all() (robot.libraries.dialogs_py.SelectionDialog method), 137
unescape() (in module robot.utils.escaping), 269
unescape() (robot.parsing.htmlreader.HtmlReader method), 191
unescape() (robot.utils.escaping.Unescaper method), 269
Unescaper (class in robot.utils.escaping), 269
unic() (in module robot.utils.unic), 279
unknown_decl() (robot.parsing.htmlreader.HtmlReader method), 190
unregister() (robot.output.listenermethods.LibraryListenerMethods method), 182
unregister() (robot.output.listeners.LibraryListeners method), 182

366 Index
unregister_console_logger() (robot.output.logger.Logger method), 183
unregister_logger() (robot.output.logger.Logger method), 183
unregister_xml_logger() (robot.output.logger.Logger method), 183
unstrip() (robot.libraries.XML.NameSpaceStripper method), 102
update() (robot.libraries.dialogs_py.InputDialog method), 124
update() (robot.libraries.dialogs_py.MessageDialog method), 111
update() (robot.libraries.dialogs_py.PassFailDialog method), 150
update() (robot.libraries.dialogs_py.SelectionDialog method), 137
update() (robot.model.metadata.Metadata method), 163
update() (robot.utils.dotdict.DotDict method), 268
update() (robot.utils.normalizing.NormalizedDict method), 275
update() (robot.utils.ordereddict.OrderedDict method), 275
update() (robot.variables.scopes.GlobalVariables method), 283
update() (robot.variables.scopes.SetVariables method), 284
update() (robot.variables.store.VariableStore method), 284
update() (robot.variables.variables.Variables method), 285
update_idletasks() (robot.libraries.dialogs_py.InputDialog method), 124
update_idletasks() (robot.libraries.dialogs_py.MessageDialog method), 111
update_idletasks() (robot.libraries.dialogs_py.PassFailDialog method), 150
update_idletasks() (robot.libraries.dialogs_py.SelectionDialog method), 137
updatepos() (robot.parsing.htmlreader.HtmlReader method), 191
user_agent (robot.libraries.Remote.TimeoutTransport attribute), 74
user_keywords (robot.running.model.TestSuite attribute), 252
UserErrorHandler (class in robot.running.usererrorhandler), 260
UserKeyword (class in robot.parsing.tablepopulators), 203
UserKeywordRunner (class in robot.running.userkeywordrunner), 261
UserLibrary (class in robot.running.userkeyword), 261
Utf8Reader (class in robot.utils.utf8reader), 280

V
validate() (robot.libdoc.LibDoc method), 293
validate() (robot.robot.Rebot method), 295
validate() (robot.run.RobotFramework method), 296
validate() (robot.running.arguments.argumentvalidator.ArgumentValidator method), 241
validate() (robot.testdoc.TestDoc method), 298
validate() (robot.tidy.TidyCommandLine method), 300
validate() (robot.utils.application.Application method), 262
validate() (robot.variables.assigner.AssignmentValidator method), 280
validate_assignment() (robot.variables.assigner.VariableAssignment method), 280
validate_command() (robot.libdocpkg.consoleviewer.ConsoleViewer class method), 20
validate_flatten_keyword() (in module robot.result.flattenkeywordmatcher), 215
validate_var() (in module robot.variables.isvar), 282
value (robot.parsing.comments.Comments attribute), 189
values (robot.running.model.ForLoop attribute), 249
values() (robot.model.metadata.Metadata method), 163
values() (robot.running.importer.ImportCache method), 246
values() (robot.utils.dotdict.DotDict method), 268
values() (robot.utils.normalizing.NormalizedDict method), 275
values() (robot.utils.ordereddict.OrderedDict method), 275
VarargsHandler (class in robot.running.arguments.javaargumentcoercer), 242
Variable (class in robot.parsing.model), 195
Variable (class in robot.running.model), 255
variable_files (robot.conf.settings.RobotSettings attribute), 17
variable_not_found() (in module robot.variables.notfound), 282
variable_should_exist() (robot.libraries.BuiltIn.BuiltIn method), 43
variable_should_not_exist() (robot.libraries.BuiltIn.BuiltIn method), 43
VariableAssigner (class in robot.variables.assigner), 280
VariableAssignment (class in robot.variables.assigner), 280
VariableError, 289
visit_keyword() (robot.model.totalstatistics.TotalStatisticsBuilder method), 174
visit_keyword() (robot.model.visitor.SuiteVisitor method), 176
visit_keyword() (robot.output.console.dotted.StatusReporter method), 178
visit_keyword() (robot.output.xmllogger.XmlLogger method), 188
visit_keyword() (robot.reporting.outputwriter.OutputWriter method), 206
visit_keyword() (robot.reporting.xunitwriter.XUnitFileWriter method), 209
visit_keyword() (robot.result.configurer.SuiteConfigurer method), 212
visit_keyword() (robot.result.keywordremover.AllKeywordsRemover method), 207
visit_keyword() (robot.result.keywordremover.ByNameKeywordRemover method), 217
visit_keyword() (robot.result.keywordremover.ByTagKeywordRemover method), 218
visit_keyword() (robot.result.keywordremover.ForLoopItemsRemover method), 219
visit_keyword() (robot.result.keywordremover.PassedKeywordRemover method), 217
visit_keyword() (robot.result.keywordremover.WaitUntilKeywordSucceedsRemover method), 220
visit_keyword() (robot.result.keywordremover.WarningAndErrorFinder method), 220
visit_keyword() (robot.result.merger.Merger method), 222
visit_keyword() (robot.result.messagefilter.MessageFilter method), 223
visit_keyword() (robot.result.resultbuilder.RemoveKeywords method), 231
visit_keyword() (robot.result.suiteteardownfailed.SuiteTeardownFailed method), 232
visit_keyword() (robot.result.suiteteardownfailed.SuiteTeardownFailureHandler method), 231
visit_keyword() (robot.result.visitor.ResultVisitor method), 234
visit_keyword() (robot.running.randomizer.Randomizer method), 256
visit_keyword() (robot.running.runner.Runner method), 257
visit_message() (robot.conf.gatherfailed.GatherFailedSuites method), 16
visit_message() (robot.conf.gatherfailed.GatherFailedTests method), 15
visit_message() (robot.model.configurer.SuiteConfigurer method), 156
visit_message() (robot.model.filter.EmptySuiteRemover method), 157
visit_message() (robot.model.filter.Filter method), 158
visit_message() (robot.model.modifier.ModelModifier method), 164
visit_message() (robot.model.statistics.StatisticsBuilder method), 165
visit_message() (robot.model.tagsetter.TagSetter method), 169
visit_message() (robot.model.totalstatistics.TotalStatisticsBuilder method), 175
visit_message() (robot.model.visitor.SuiteVisitor method), 176
visit_message() (robot.output.console.dotted.StatusReporter method), 178
visit_message() (robot.output.xmllogger.XmlLogger method), 188
visit_message() (robot.reporting.outputwriter.OutputWriter method), 207
visit_message() (robot.reporting.xunitwriter.XUnitFileWriter method), 210
visit_message() (robot.result.configurer.SuiteConfigurer method), 212
visit_message() (robot.result.keywordremover.AllKeywordsRemover method), 216
visit_message() (robot.result.keywordremover.ByNameKeywordRemover method), 217
visit_message() (robot.result.keywordremover.ByTagKeywordRemover method), 218
visit_message() (robot.result.keywordremover.ForLoopItemsRemover method), 219
visit_message() (robot.result.keywordremover.PassedKeywordRemover method), 217
visit_message() (robot.result.keywordremover.WaitUntilKeywordSucceedsRemover method), 220
visit_message() (robot.result.keywordremover.WarningAndErrorFinder method), 220
visit_message() (robot.result.merger.Merger method), 222
visit_message() (robot.result.messagefilter.MessageFilter method), 223
visit_message() (robot.result.resultbuilder.RemoveKeywords method), 231
visit_message() (robot.result.suiteteardownfailed.SuiteTeardownFailed method), 232
visit_message() (robot.result.suiteteardownfailed.SuiteTeardownFailureHandler method), 231
visit_message() (robot.result.visitor.ResultVisitor method), 234
visit_message() (robot.running.randomizer.Randomizer method), 257
visit_message() (robot.running.runner.Runner method), 258
visit_result() (robot.output.xmllogger.XmlLogger method), 188
visit_result() (robot.reporting.outputwriter.OutputWriter method), 207
visit_result() (robot.reporting.xunitwriter.XUnitFileWriter method), 210
Robot Framework Documentation, Release 3.1.0.dev1

visit_result() (robot.result.visitor.ResultVisitor method), 233
visit_stat() (robot.result.visitor.ResultVisitor method), 210
visit_stat() (robot.result.visitor.ResultVisitor method), 234
visit_statistics() (robot.output.xmllogger.XmlLogger method), 188
visit_statistics() (robot.reporting.outputwriter.OutputWriter method), 207
visit_statistics() (robot.reporting.xmllogger.XmlLogger method), 188
visit_suite() (robot.conf.gatherfailed.GatherFailedSuites method), 16
visit_suite() (robot.conf.gatherfailed.GatherFailedTests method), 15
visit_suite() (robot.model.configurer.SuiteConfigurer method), 155
visit_suite() (robot.model.filter.EmptySuiteRemover method), 157
visit_suite() (robot.model.filter.Filter method), 158
visit_suite() (robot.model.modifier.ModelModifier method), 163
visit_suite() (robot.model.statistics.StatisticsBuilder method), 165
visit_suite() (robot.model.tagsetter.TagSetter method), 169
visit_suite() (robot.model.totalstatistics.TotalStatisticsBuilder method), 175
visit_suite() (robot.model.visitor.SuiteVisitor method), 176
visit_suite() (robot.output.console.dotted.StatusReporter method), 178
visit_suite() (robot.output.xmllogger.XmlLogger method), 188
visit_suite() (robot.reporting.outputwriter.OutputWriter method), 207
visit_suite() (robot.reporting.xunitwriter.XUnitFileWriter method), 210
visit_suite() (robot.result.configurer.SuiteConfigurer method), 212
visit_suite() (robot.result.keywordremover.AllKeywordsRemover method), 216
visit_suite() (robot.result.keywordremover.ByNameKeywordRemover method), 217
visit_suite() (robot.result.keywordremover.ByTagKeywordRemover method), 218
visit_suite() (robot.result.keywordremover.ForLoopItemsRemover method), 219
visit_suite() (robot.result.keywordremover.PassedKeywordRemover method), 217
visit_suite() (robot.result.keywordremover.WaitUntilKeywordSucceedsRemover method), 220
visit_suite() (robot.result.keywordremover.WarningAndErrorFinder method), 221
visit_suite() (robot.result.merger.Merger method), 222
visit_suite() (robot.result.messagefilter.MessageFilter method), 223
visit_suite() (robot.result.resultbuilder.RemoveKeywords method), 231
visit_suite() (robot.result.suiteteardownfailed.SuiteTeardownFailed method), 232
visit_suite() (robot.result.suiteteardownfailed.SuiteTeardownFailureHandler method), 232
visit_suite() (robot.running.randomizer.Randomizer method), 257
visit_suite() (robot.running.runner.Runner method), 258
visit_suite_stats() (robot.output.xmllogger.XmlLogger method), 188
visit_suite_stats() (robot.reporting.outputwriter.OutputWriter method), 207
visit_suite_stats() (robot.reporting.xunitwriter.XUnitFileWriter method), 210
visit_suite_stats() (robot.result.visitor.ResultVisitor method), 234
visit_test() (robot.conf.gatherfailed.GatherFailedSuites method), 15
visit_test() (robot.conf.gatherfailed.GatherFailedTests method), 15
visit_test() (robot.model.configurer.SuiteConfigurer method), 156
visit_test() (robot.model.filter.EmptySuiteRemover method), 156
visit_test() (robot.model.modifier.ModelModifier method), 158
visit_test() (robot.model.statistics.StatisticsBuilder method), 164
visit_test() (robot.model.tagsetter.TagSetter method), 169
visit_test() (robot.result.keywordremover.AllKeywordsRemover method), 164
visit_test() (robot.result.keywordremover.ByNameKeywordRemover method), 165
visit_test() (robot.result.keywordremover.ByTagKeywordRemover method), 169
visit_test() (robot.result.keywordremover.ForLoopItemsRemover method), 174

370 Index
visit_test() (robot.model.visitor.SuiteVisitor method), 176
visit_test() (robot.output.console.dotted.StatusReporter method), 177
visit_test() (robot.output.xmllogger.XmlLogger method), 188
visit_test() (robot.reporting.outputwriter.OutputWriter method), 208
visit_test() (robot.reporting.xunitwriter.XUnitFileWriter method), 209
visit_test() (robot.result.configurer.SuiteConfigurer method), 212
visit_test() (robot.result.keywordremover.AllKeywordsRemover method), 216
visit_test() (robot.result.keywordremover.ByNameKeywordRemover method), 218
visit_test() (robot.result.keywordremover.ByTagKeywordRemover method), 219
visit_test() (robot.result.keywordremover.ForLoopItemsRemover method), 219
visit_test() (robot.result.keywordremover.PassedKeywordRemover method), 216
visit_test() (robot.result.keywordremover.WaitUntilKeywordSucceedsRemover method), 220
visit_test() (robot.result.keywordremover.ResultVisitor method), 221
visit_test() (robot.running.randomizer.Randomizer method), 222
visit_test() (robot.running.runner.Runner method), 226
visit_total_statistics() (robot.output.xmllogger.XmlLogger method), 188
visit_total_statistics() (robot.reporting.outputwriter.OutputWriter method), 208
visit_total_statistics() (robot.reporting.xunitwriter.XUnitFileWriter method), 210
visit_total_statistics() (robot.result.visitor.ResultVisitor method), 233

wait_until_keyword_succeeds() (robot.libraries.BuiltIn.BuiltIn method), 44
wait_until_removed() (robot.libraries.OperatingSystem.OperatingSystem method), 60
wait_variable() (robot.librariesdialogs_py.InputDialog method), 124
wait_variable() (robot.librariesdialogs_py.MessageDialog method), 111
wait_variable() (robot.librariesdialogs_py.PassFailDialog method), 150
wait_variable() (robot.librariesdialogs_py.SelectionDialog method), 137
wait_variable() (robot.librariesdialogs_py.SuiteVisitor method), 235
wait_variable() (robot.libraries.dialogs_py.InputDialog method), 124
wait_variable() (robot.libraries.dialogs_py.MessageDialog method), 111
wait_variable() (robot.libraries.dialogs_py.PassFailDialog method), 150
wait_variable() (robot.libraries.dialogs_py.SelectionDialog method), 137
wait_window() (robot.libraries.dialogs_py.MessageDialog method), 111
wait_window() (robot.libraries.dialogs_py.PassFailDialog method), 150
wait_window() (robot.libraries.dialogs_py.SelectionDialog method), 137
WaitUntilKeywordSucceedsRemover (class in robot.result.keywordremover), 219
waitvar() (robot.libraries.dialogs_py.InputDialog method), 124
waitvar() (robot.libraries.dialogs_py.MessageDialog method), 111
waitvar() (robot.libraries.dialogs_py.PassFailDialog method), 150
waitvar() (robot.libraries.dialogs_py.SelectionDialog method), 137
warn() (in module robot.api.logger), 14
warn() (in module robot.output.librarylogger), 180
warn() (robot.output.filelogger.FileLogger method), 180
warn() (robot.output.logger.Logger method), 184
warn() (robot.output.loggerhelper.AbstractLogger method), 184
warn() (robot.output.output.Output method), 185
warning() (robot.parsing.restsupport.CaptureRobotData method), 198
WarningAndErrorFinder (class in robot.result.keywordremover), 220
winfo_atom() (robot.libraries.dialogs_py.InputDialog method), 124
winfo_atom() (robot.libraries.dialogs_py.MessageDialog method), 111
winfo_atom() (robot.libraries.dialogs_py.PassFailDialog method), 150
winfo_atom() (robot.libraries.dialogs_py.SelectionDialog method), 137

Index 371
winfo_atom() (robot.libraries.dialogs_py.PassFailDialog method), 151
winfo_atom() (robot.libraries.dialogs_py.SelectionDialog method), 138
winfo_atomname() (robot.libraries.dialogs_py.InputDialog method), 125
winfo_atomname() (robot.libraries.dialogs_py.MessageDialog method), 112
winfo_atomname() (robot.libraries.dialogs_py.PassFailDialog method), 151
winfo_atomname() (robot.libraries.dialogs_py.SelectionDialog method), 138
winfo_cells() (robot.libraries.dialogs_py.InputDialog method), 125
winfo_cells() (robot.libraries.dialogs_py.MessageDialog method), 112
winfo_cells() (robot.libraries.dialogs_py.PassFailDialog method), 151
winfo_cells() (robot.libraries.dialogs_py.SelectionDialog method), 138
winfo_children() (robot.libraries.dialogs_py.InputDialog method), 125
winfo_children() (robot.libraries.dialogs_py.MessageDialog method), 112
winfo_children() (robot.libraries.dialogs_py.PassFailDialog method), 151
winfo_children() (robot.libraries.dialogs_py.SelectionDialog method), 138
winfo_class() (robot.libraries.dialogs_py.InputDialog method), 125
winfo_class() (robot.libraries.dialogs_py.MessageDialog method), 112
winfo_class() (robot.libraries.dialogs_py.PassFailDialog method), 151
winfo_class() (robot.libraries.dialogs_py.SelectionDialog method), 138
winfo_colormapfull() (robot.libraries.dialogs_py.InputDialog method), 125
winfo_colormapfull() (robot.libraries.dialogs_py.MessageDialog method), 112
winfo_colormapfull() (robot.libraries.dialogs_py.PassFailDialog method), 151
winfo_colormapfull() (robot.libraries.dialogs_py.SelectionDialog method), 138
winfo_depth() (robot.libraries.dialogs_py.MessageDialog method), 112
winfo_depth() (robot.libraries.dialogs_py.PassFailDialog method), 151
winfo_depth() (robot.libraries.dialogs_py.SelectionDialog method), 138
winfo_exists() (robot.libraries.dialogs_py.InputDialog method), 125
winfo_exists() (robot.libraries.dialogs_py.MessageDialog method), 112
winfo_exists() (robot.libraries.dialogs_py.PassFailDialog method), 151
winfo_exists() (robot.libraries.dialogs_py.SelectionDialog method), 138
winfo_fpixels() (robot.libraries.dialogs_py.InputDialog method), 125
winfo_fpixels() (robot.libraries.dialogs_py.MessageDialog method), 112
winfo_fpixels() (robot.libraries.dialogs_py.PassFailDialog method), 151
winfo_fpixels() (robot.libraries.dialogs_py.SelectionDialog method), 138
winfo_geometry() (robot.libraries.dialogs_py.InputDialog method), 125
winfo_geometry() (robot.libraries.dialogs_py.MessageDialog method), 112
winfo_geometry() (robot.libraries.dialogs_py.PassFailDialog method), 151
winfo_geometry() (robot.libraries.dialogs_py.SelectionDialog method), 138
winfo_height() (robot.libraries.dialogs_py.InputDialog method), 125
winfo_height() (robot.libraries.dialogs_py.MessageDialog method), 112
winfo_height() (robot.libraries.dialogs_py.PassFailDialog method), 151
winfo_height() (robot.libraries.dialogs_py.SelectionDialog method), 138
winfo_id() (robot.libraries.dialogs_py.InputDialog method), 125
winfo_id() (robot.libraries.dialogs_py.MessageDialog method), 112
winfo_id() (robot.libraries.dialogs_py.PassFailDialog method), 151
winfo_id() (robot.libraries.dialogs_py.SelectionDialog method), 138
winfo_interps() (robot.libraries.dialogs_py.InputDialog method), 125
winfo_interps() (robot.libraries.dialogs_py.MessageDialog method), 112
winfo_interps() (robot.libraries.dialogs_py.PassFailDialog method), 151
winfo_interps() (robot.libraries.dialogs_py.SelectionDialog method), 138
winfo_interps() (robot.libraries.dialogs_py.MessageDialog method), 112
winfo_interps() (robot.libraries.dialogs_py.PassFailDialog method), 151
winfo_interps() (robot.libraries.dialogs_py.SelectionDialog method), 138
winfo_interps() (robot.libraries.dialogs_py.MessageDialog method), 112
winfo_interps() (robot.libraries.dialogs_py.PassFailDialog method), 151
winfo_interps() (robot.libraries.dialogs_py.SelectionDialog method), 138
winfo_interps() (robot.libraries.dialogs_py.MessageDialog method), 112
winfo_interps() (robot.libraries.dialogs_py.PassFailDialog method), 151
winfo_interps() (robot.libraries.dialogs_py.SelectionDialog method), 138
winfo_interps() (robot.libraries.dialogs_py.MessageDialog method), 112
winfo_interps() (robot.libraries.dialogs_py.PassFailDialog method), 151
winfo_interps() (robot.libraries.dialogs_py.SelectionDialog method), 138

winfo_ismapped() (robot.libraries.dialogs_py.InputDialog method), 125
winfo_ismapped() (robot.libraries.dialogs_py.MessageDialog method), 112
winfo_ismapped() (robot.libraries.dialogs_py.PassFailDialog method), 151
winfo_ismapped() (robot.libraries.dialogs_py.SelectionDialog method), 138
winfo_manager() (robot.libraries.dialogs_py.InputDialog method), 125
winfo_manager() (robot.libraries.dialogs_py.MessageDialog method), 112
winfo_manager() (robot.libraries.dialogs_py.PassFailDialog method), 151
winfo_manager() (robot.libraries.dialogs_py.SelectionDialog method), 138
winfo_name() (robot.libraries.dialogs_py.InputDialog method), 125
winfo_name() (robot.libraries.dialogs_py.MessageDialog method), 112
winfo_name() (robot.libraries.dialogs_py.PassFailDialog method), 151
winfo_name() (robot.libraries.dialogs_py.SelectionDialog method), 138
winfo_parent() (robot.libraries.dialogs_py.InputDialog method), 125
winfo_parent() (robot.libraries.dialogs_py.MessageDialog method), 112
winfo_parent() (robot.libraries.dialogs_py.PassFailDialog method), 151
winfo_parent() (robot.libraries.dialogs_py.SelectionDialog method), 138
winfo_pathname() (robot.libraries.dialogs_py.InputDialog method), 125
winfo_pathname() (robot.libraries.dialogs_py.MessageDialog method), 112
winfo_pathname() (robot.libraries.dialogs_py.PassFailDialog method), 151
winfo_pathname() (robot.libraries.dialogs_py.SelectionDialog method), 138
winfo_pixels() (robot.libraries.dialogs_py.InputDialog method), 125
winfo_pixels() (robot.libraries.dialogs_py.MessageDialog method), 112
winfo_pixels() (robot.libraries.dialogs_py.PassFailDialog method), 151
winfo_pixels() (robot.libraries.dialogs_py.SelectionDialog method), 138
winfo_pointerx() (robot.libraries.dialogs_py.InputDialog method), 125
winfo_pointerx() (robot.libraries.dialogs_py.MessageDialog method), 112
winfo_pointerx() (robot.libraries.dialogs_py.PassFailDialog method), 151
winfo_pointerx() (robot.libraries.dialogs_py.SelectionDialog method), 138
winfo_pointery() (robot.libraries.dialogs_py.InputDialog method), 126
winfo_pointery() (robot.libraries.dialogs_py.MessageDialog method), 113
winfo_pointery() (robot.libraries.dialogs_py.PassFailDialog method), 152
winfo_pointery() (robot.libraries.dialogs_py.SelectionDialog method), 139
winfo_reqheight() (robot.libraries.dialogs_py.InputDialog method), 126
winfo_reqheight() (robot.libraries.dialogs_py.MessageDialog method), 113
winfo_reqheight() (robot.libraries.dialogs_py.PassFailDialog method), 152
winfo_reqheight() (robot.libraries.dialogs_py.SelectionDialog method), 139
winfo_reqwidth() (robot.libraries.dialogs_py.InputDialog method), 126
winfo_reqwidth() (robot.libraries.dialogs_py.MessageDialog method), 113
winfo_reqwidth() (robot.libraries.dialogs_py.PassFailDialog method), 152
winfo_reqwidth() (robot.libraries.dialogs_py.SelectionDialog method), 139
winfo_rgb() (robot.libraries.dialogs_py.InputDialog method), 126
winfo_rgb() (robot.libraries.dialogs_py.MessageDialog method), 113
winfo_rgb() (robot.libraries.dialogs_py.PassFailDialog method), 152
winfo_rgb() (robot.libraries.dialogs_py.SelectionDialog method), 139
winfo_rootx() (robot.libraries.dialogs_py.InputDialog method), 126
winfo_rootx() (robot.libraries.dialogs_py.MessageDialog method), 113
winfo_rootx() (robot.libraries.dialogs_py.PassFailDialog method), 152
winfo_rootx() (robot.libraries.dialogs_py.SelectionDialog method), 139
winfo_rooty() (robot.libraries.dialogs_py.InputDialog method), 126
winfo_rooty() (robot.libraries.dialogs_py.MessageDialog method), 113
winfo_rooty() (robot.libraries.dialogs_py.PassFailDialog method), 152
winfo_rooty() (robot.libraries.dialogs_py.SelectionDialog method), 139

Index 373
winfo_rooty() (robot.libraries.dialogs_py.PassFailDialog
method), 152
winfo_rooty() (robot.libraries.dialogs_py.SelectionDialog
method), 139
winfo_screen() (robot.libraries.dialogs_py.InputDialog
method), 126
winfo_screen() (robot.libraries.dialogs_py.MessageDialog
method), 113
winfo_screen() (robot.libraries.dialogs_py.PassFailDialog
method), 152
winfo_screen() (robot.libraries.dialogs_py.SelectionDialog
method), 139
winfo_screencells() (robot.libraries.dialogs_py.InputDialog
method), 126
winfo_screencells() (robot.libraries.dialogs_py.MessageDialog
method), 113
winfo_screencells() (robot.libraries.dialogs_py.PassFailDialog
method), 152
winfo_screencells() (robot.libraries.dialogs_py.SelectionDialog
method), 139
winfo_screendepth() (robot.libraries.dialogs_py.InputDialog
method), 126
winfo_screendepth() (robot.libraries.dialogs_py.MessageDialog
method), 113
winfo_screendepth() (robot.libraries.dialogs_py.PassFailDialog
method), 152
winfo_screendepth() (robot.libraries.dialogs_py.SelectionDialog
method), 139
winfo_screenheight() (robot.libraries.dialogs_py.InputDialog
method), 126
winfo_screenheight() (robot.libraries.dialogs_py.MessageDialog
method), 113
winfo_screenheight() (robot.libraries.dialogs_py.PassFailDialog
method), 152
winfo_screenheight() (robot.libraries.dialogs_py.SelectionDialog
method), 139
winfo_screenmmheight() (robot.libraries.dialogs_py.InputDialog
method), 126
winfo_screenmmheight() (robot.libraries.dialogs_py.MessageDialog
method), 113
winfo_screenmmheight() (robot.libraries.dialogs_py.PassFailDialog
method), 152
winfo_screenmmheight() (robot.libraries.dialogs_py.SelectionDialog
method), 139
winfo_screenmmwidth() (robot.libraries.dialogs_py.InputDialog
method), 126
winfo_screenmmwidth() (robot.libraries.dialogs_py.MessageDialog
method), 113
winfo_screenmmwidth() (robot.libraries.dialogs_py.PassFailDialog
method), 152
winfo_screenmmwidth() (robot.libraries.dialogs_py.SelectionDialog
method), 139
winfo_screenwidth() (robot.libraries.dialogs_py.InputDialog
method), 126
winfo_screenwidth() (robot.libraries.dialogs_py.MessageDialog
method), 113
winfo_screenwidth() (robot.libraries.dialogs_py.PassFailDialog
method), 152
winfo_screenwidth() (robot.libraries.dialogs_py.SelectionDialog
method), 139
winfo_server() (robot.libraries.dialogs_py.InputDialog
method), 126
winfo_server() (robot.libraries.dialogs_py.MessageDialog
method), 113
winfo_server() (robot.libraries.dialogs_py.PassFailDialog
method), 152
winfo_server() (robot.libraries.dialogs_py.SelectionDialog
method), 139
winfo_toplevel() (robot.libraries.dialogs_py.InputDialog
method), 126
winfo_toplevel() (robot.libraries.dialogs_py.MessageDialog
method), 113
winfo_toplevel() (robot.libraries.dialogs_py.PassFailDialog
method), 152
winfo_toplevel() (robot.libraries.dialogs_py.SelectionDialog
method), 139
winfo_viewable() (robot.libraries.dialogs_py.InputDialog
method), 126
winfo_viewable() (robot.libraries.dialogs_py.MessageDialog
method), 113
winfo_viewable() (robot.libraries.dialogs_py.PassFailDialog
method), 152
winfo_viewable() (robot.libraries.dialogs_py.SelectionDialog
method), 139
winfo_visual() (robot.libraries.dialogs_py.InputDialog
method), 126
winfo_visual() (robot.libraries.dialogs_py.MessageDialog
method), 113
winfo_visual() (robot.libraries.dialogs_py.PassFailDialog
method), 152
winfo_visual() (robot.libraries.dialogs_py.SelectionDialog
method), 139
winfo_visualid() (robot.libraries.dialogs_py.InputDialog
method), 126
winfo_visualid() (robot.libraries.dialogs_py.MessageDialog
method), 113
winfo_visualid() (robot.libraries.dialogs_py.PassFailDialog
method), 152
winfo_visualid() (robot.libraries.dialogs_py.SelectionDialog
method), 139
winfo_visualid() (robot.libraries.dialogs_py.PassFailDialog method), 152
winfo_visualid() (robot.libraries.dialogs_py.SelectionDialog method), 139
winfo_visualsavailable() (robot.libraries.dialogs_py.InputDialog method), 126
winfo_visualsavailable() (robot.libraries.dialogs_py.MessageDialog method), 113
winfo_visualsavailable() (robot.libraries.dialogs_py.PassFailDialog method), 152
winfo_visualsavailable() (robot.libraries.dialogs_py.SelectionDialog method), 139
winfo_vrootheight() (robot.libraries.dialogs_py.InputDialog method), 127
winfo_vrootheight() (robot.libraries.dialogs_py.MessageDialog method), 114
winfo_vrootheight() (robot.libraries.dialogs_py.PassFailDialog method), 153
winfo_vrootheight() (robot.libraries.dialogs_py.SelectionDialog method), 140
winfo_vrootwidth() (robot.libraries.dialogs_py.InputDialog method), 127
winfo_vrootwidth() (robot.libraries.dialogs_py.MessageDialog method), 114
winfo_vrootwidth() (robot.libraries.dialogs_py.PassFailDialog method), 153
winfo_vrootwidth() (robot.libraries.dialogs_py.SelectionDialog method), 140
winfo_vrootx() (robot.libraries.dialogs_py.InputDialog method), 127
winfo_vrootx() (robot.libraries.dialogs_py.MessageDialog method), 114
winfo_vrootx() (robot.libraries.dialogs_py.PassFailDialog method), 153
winfo_vrootx() (robot.libraries.dialogs_py.SelectionDialog method), 140
winfo_vrooty() (robot.libraries.dialogs_py.InputDialog method), 127
winfo_vrooty() (robot.libraries.dialogs_py.MessageDialog method), 114
winfo_vrooty() (robot.libraries.dialogs_py.PassFailDialog method), 153
winfo_vrooty() (robot.libraries.dialogs_py.SelectionDialog method), 140
winfo_width() (robot.libraries.dialogs_py.InputDialog method), 127
winfo_width() (robot.libraries.dialogs_py.MessageDialog method), 114
winfo_width() (robot.libraries.dialogs_py.PassFailDialog method), 153
winfo_width() (robot.libraries.dialogs_py.SelectionDialog method), 140
winfo_x() (robot.libraries.dialogs_py.MessageDialog method), 114
winfo_x() (robot.libraries.dialogs_py.PassFailDialog method), 153
winfo_x() (robot.libraries.dialogs_py.SelectionDialog method), 140
wm_aspect() (robot.libraries.dialogs_py.InputDialog method), 127
wm_aspect() (robot.libraries.dialogs_py.MessageDialog method), 114
wm_aspect() (robot.libraries.dialogs_py.PassFailDialog method), 153
wm_aspect() (robot.libraries.dialogs_py.SelectionDialog method), 140
wm_attributes() (robot.libraries.dialogs_py.InputDialog method), 127
wm_attributes() (robot.libraries.dialogs_py.MessageDialog method), 114
wm_attributes() (robot.libraries.dialogs_py.PassFailDialog method), 153
wm_attributes() (robot.libraries.dialogs_py.SelectionDialog method), 140
wm_client() (robot.libraries.dialogs_py.InputDialog method), 127
wm_client() (robot.libraries.dialogs_py.MessageDialog method), 114
wm_client() (robot.libraries.dialogs_py.PassFailDialog method), 153
wm_client() (robot.libraries.dialogs_py.SelectionDialog method), 140
wm_colormapwindows() (robot.libraries.dialogs_py.InputDialog method), 127
wm_colormapwindows() (robot.libraries.dialogs_py.MessageDialog method), 114
wm_colormapwindows() (robot.libraries.dialogs_py.PassFailDialog method), 153
wm_colormapwindows() (robot.libraries.dialogs_py.SelectionDialog method), 140
(robot.libraries.dialogs_py.PassFailDialog method), 153
wm_colormapwindows()
(robot.libraries.dialogs_py.SelectionDialog method), 140
wm_command() (robot.libraries.dialogs_py.InputDialog method), 127
wm_command() (robot.libraries.dialogs_py.MessageDialog method), 114
wm_command() (robot.libraries.dialogs_py.PassFailDialog method), 153
wm_command() (robot.libraries.dialogs_py.SelectionDialog method), 140
wm_deiconify() (robot.libraries.dialogs_py.InputDialog method), 127
wm_deiconify() (robot.libraries.dialogs_py.MessageDialog method), 114
wm_deiconify() (robot.libraries.dialogs_py.PassFailDialog method), 153
wm_deiconify() (robot.libraries.dialogs_py.SelectionDialog method), 140
wm_focusselmodel() (robot.libraries.dialogs_py.InputDialog method), 128
wm_focusselmodel() (robot.libraries.dialogs_py.MessageDialog method), 115
wm_focusselmodel() (robot.libraries.dialogs_py.PassFailDialog method), 154
wm_focusselmodel() (robot.libraries.dialogs_py.SelectionDialog method), 141
wm_focus() (robot.libraries.dialogs_py.InputDialog method), 128
wm_focus() (robot.libraries.dialogs_py.MessageDialog method), 115
wm_focus() (robot.libraries.dialogs_py.PassFailDialog method), 154
wm_focus() (robot.libraries.dialogs_py.SelectionDialog method), 141
wm_iconbitmap() (robot.libraries.dialogs_py.InputDialog method), 128
wm_iconbitmap() (robot.libraries.dialogs_py.MessageDialog method), 115
wm_iconbitmap() (robot.libraries.dialogs_py.PassFailDialog method), 154
wm_iconbitmap() (robot.libraries.dialogs_py.SelectionDialog method), 141
wm_iconify() (robot.libraries.dialogs_py.InputDialog method), 128
wm_iconify() (robot.libraries.dialogs_py.MessageDialog method), 115
wm_iconify() (robot.libraries.dialogs_py.PassFailDialog method), 154
wm_iconify() (robot.libraries.dialogs_py.SelectionDialog method), 141
wm_iconmask() (robot.libraries.dialogs_py.InputDialog method), 128
wm_iconmask() (robot.libraries.dialogs_py.MessageDialog method), 115
wm_iconmask() (robot.libraries.dialogs_py.PassFailDialog method), 154
wm_iconmask() (robot.libraries.dialogs_py.SelectionDialog method), 141
wm_iconname() (robot.libraries.dialogs_py.InputDialog method), 128
wm_iconname() (robot.libraries.dialogs_py.MessageDialog method), 115
wm_iconname() (robot.libraries.dialogs_py.PassFailDialog method), 154
wm_iconname() (robot.libraries.dialogs_py.SelectionDialog method), 141
wm_iconposition() (robot.libraries.dialogs_py.InputDialog method), 128
wm_iconposition() (robot.libraries.dialogs_py.MessageDialog method), 115
wm_iconposition() (robot.libraries.dialogs_py.PassFailDialog method), 154
wm_iconposition() (robot.libraries.dialogs_py.SelectionDialog method), 141
wm_iconwindow() (robot.libraries.dialogs_py.InputDialog method), 128
wm_iconwindow() (robot.libraries.dialogs_py.MessageDialog method), 115
wm_iconwindow() (robot.libraries.dialogs_py.PassFailDialog method), 154
wm_iconwindow() (robot.libraries.dialogs_py.SelectionDialog method), 141

Index
Index 377
write() (robot.libdocpkg.htmlwriter.LibdocModelWriter method), 21
write() (robot.libdocpkg.xmlwriter.LibdocXmlWriter method), 22
write() (robot.libraries.Telnet.TelnetConnection method), 87
write() (robot.output.console.highlighting.HighlightingStream method), 178
write() (robot.output.filelogger.FileLogger method), 180
write() (robot.output.logger.Logger method), 184
write() (robot.output.loggerhelper.AbstractLogger method), 184
write() (robot.output.output.Output method), 185
write() (robot.reporting.jswriter.JsResultWriter method), 206
write() (robot.reporting.jswriter.SplitLogWriter method), 206
write() (robot.reporting.jswriter.SuiteWriter method), 206
write() (robot.reporting.logreportwriters.LogWriter method), 206
write() (robot.reporting.logreportwriters.RobotModelWriter method), 206
write() (robot.reporting.xmlelementhandlers.XmlElementHandler method), 235
write() (robot.reporting.xunitwriter.XUnitWriter method), 209
write_bare() (robot.libraries.Telnet.TelnetConnection method), 88
write_control_character() (robot.libraries.Telnet.TelnetConnection method), 88
write_data() (robot.libdocpkg.htmlwriter.LibdocModelWriter method), 21
write_data() (robot.testdoc.TestdocModelWriter method), 299
write_json() (robot.htmldata.jsonwriter.JsonWriter method), 19
write_results() (robot.reporting.resultwriter.ResultWriter method), 208
write_until_expected_output() (robot.libraries.Telnet.TelnetConnection method), 88
WritingContext (class in robot.writer.datafilewriter), 286

X
XML (class in robot.libraries.XML), 91
xml_escape() (in module robot.utils.markuputils), 272
XmlElementHandler (class in robot.result.xmlelementhandlers), 235
XmlLogger (class in robot.output.xmllogger), 187
XmlRpcRemoteClient (class in robot.libraries.Remote), 73
XmlWriter (class in robot.utils.markupwriters), 272
xunit (robot.conf.settings.RebotSettings attribute), 18
xunit (robot.conf.settings.RobotSettings attribute), 17
xunit_skip_noncritical (robot.conf.settings.RebotSettings attribute), 18
xunit_skip_noncritical (robot.conf.settings.RobotSettings attribute), 17
XUnitFileWriter (class in robot.reporting.xunitwriter), 209
XUnitWriter (class in robot.reporting.xunitwriter), 209

Y
YamlImporter (class in robot.variables.filesetter), 281
yellow() (robot.output.console.highlighting.AnsiHighlighter method), 178
yellow() (robot.output.console.highlighting.DosHighlighter method), 179
yellow() (robot.output.console.highlighting.NoHighlighting method), 178

Index