## 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 .................................................................. 349

Python Module Index .................................................. 351
Index ......................................................................... 355
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.
CHAPTER 2

Java entry points

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

The `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.
- The `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.
- The `TestSuite` class for creating executable test suites programmatically and `TestSuiteBuilder` class for creating such suites based on existing test data on the file system.
- The `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.
- The `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.
- The `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 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, --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 **prerobtmodifier** 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=None, exit=True)
Command line execution entry point for running tests.

**Parameters**

• **arguments** – Command line options and arguments as a list of strings. Starting from RF 3.1, defaults to sys.argv[1:] if not given.

• **exit** – If True, call sys.exit with the return code denoting execution status, otherwise just return the rc. New in RF 3.0.1.

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

Example:
from robot import run_cli

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

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

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

robot.rebot(*outputs, **options)
Programmatic entry point for post-processing outputs.

Parameters

- outputs – Paths to Robot Framework output files similarly as when running the robot command on the command line.

- options – Options to configure processing outputs. 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.

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

Examples:

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

Equivalent command line usage:

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

robot.rebot_cli(arguments=None, exit=True)
Command line execution entry point for post-processing outputs.

Parameters

- arguments – Command line options and arguments as a list of strings. Starting from RF 3.1, defaults to sys.argv[1:] if not given.

- exit – If True, call sys.exit with the return code denoting execution status, otherwise just return the rc. New in RF 3.0.1.

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

Example:

from robot import rebot_cli
rebot_cli(['--name', 'Example', '--log', 'NONE', 'o1.xml', 'o2.xml'])
See also the `rebot()` function that allows setting options as keyword arguments like `name="Example"` and generally has a richer API for programmatic Rebot execution.

### 4.1.1 Subpackages

#### robot.api package

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

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

Currently exposed APIs are:

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

All of the above names can be imported like:

```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.not_keyword(func)`

Decorator to disable exposing functions or methods as keywords.

Examples:
Alternatively the automatic keyword discovery can be disabled with the `library()` decorator or by setting the `ROBOT_AUTO_KEYWORDS` attribute to a false value.

New in Robot Framework 3.2.

```
@not_keyword
def not_exposed_as_keyword():
    # ...

def exposed_as_keyword():
    # ...
```

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

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

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

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

If the automatic keyword discovery has been disabled with the `library()` decorator or by setting the `ROBOT_AUTO_KEYWORDS` attribute to a false value, this decorator is needed to mark functions or methods as keywords.

Examples:

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

@keyword('Login as user "$\{user\}$" with password "$\{password\}"',
         tags=['custom name', 'embedded arguments', 'tags'])
def login(user, password):
    # ...

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

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

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

```
robot.api.deco.library(scope=None, version=None, doc_format=None, listener=None,
                       auto_keywords=False)
```

Class decorator to control keyword discovery and other library settings.

By default disables automatic keyword detection by setting class attribute `ROBOT_AUTO_KEYWORDS = False` to the decorated library. In that mode only methods decorated explicitly with the `keyword()` decorator become keywords. If that is not desired, automatic keyword discovery can be enabled by using `auto_keywords=True`.

4.1. robot package
Arguments `scope`, `version`, `doc_format` and `listener` set the library scope, version, documentation format and listener by using class attributes `ROBOT_LIBRARY_SCOPE`, `ROBOT_LIBRARY_VERSION`, `ROBOT_LIBRARY_DOC_FORMAT` and `ROBOT_LIBRARY_LISTENER`, respectively. These attributes are only set if the related arguments are given and they override possible existing attributes in the decorated class.

Examples:

```python
@library
class KeywordDiscovery:
    @keyword
def do_something(self):
        # ...

def not_keyword(self):
    # ...

@library(scope='GLOBAL', version='3.2')
class LibraryConfiguration:
    # ...
```

The `@library` decorator is new in Robot Framework 3.2.

**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:

```python
logger.info('My message')
```

instead of logging through the standard output like:

```python
print '*INFO* My message'
```

In addition to a programmatic interface being cleaner to use, this API has a benefit that the log messages have accurate timestamps.

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

**Log levels**

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

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

All methods that are used for writing messages to the log file have an optional html argument. If a message to be logged is supposed to be shown as HTML, this argument should be set to True. Alternatively, write() accepts a pseudo log level HTML.

Example

```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'.

robot.conf package

Implements settings for both test execution and output processing.

This package implements RobotSettings and RebotSettings classes used internally by the framework. There should be no need to use these classes externally.
This package can be considered relatively stable. Aforementioned classes are likely to be rewritten at some point to be more convenient to use. Instantiating them is not likely to change, though.

Submodules

**robot.conf.gatherfailed module**

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

```
visit_test (test)
   Implements traversing through the test and its keywords.
   Can be overridden to allow modifying the passed in test without calling start_test() or end_test() nor visiting keywords.

visit_keyword (kw)
   Implements traversing through the keyword and its child keywords.
   Can be overridden to allow modifying the passed in kw without calling start_keyword() or end_keyword() nor visiting child keywords.

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

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

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

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

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

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

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

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

visit_message (msg)
   Implements visiting the message.
   Can be overridden to allow modifying the passed in msg without calling start_message() or end_message().
```
visit_suite(suite)
Implements traversing through the suite and its direct children.
Can be overridden to allow modifying the passed in suite without calling start_suite() or end_suite() nor visiting child suites, tests or keywords (setup and teardown) at all.

class robot.conf.gatherfailed.GatherFailedSuites
Bases: robot.model.visitor.SuiteVisitor
start_suite(suite)
Called when suite starts. Default implementation does nothing.
Can return explicit False to stop visiting.
visit_test(test)
Implements traversing through the test and its keywords.
Can be overridden to allow modifying the passed in test without calling start_test() or end_test() nor visiting keywords.
visit_keyword(kw)
Implements traversing through the keyword and its child keywords.
Can be overridden to allow modifying the passed in kw without calling start_keyword() or end_keyword() nor visiting child keywords.
end_keyword(keyword)
Called when keyword ends. Default implementation does nothing.
end_message(msg)
Called when message ends. Default implementation does nothing.
end_suite(suite)
Called when suite ends. Default implementation does nothing.
end_test(test)
Called when test ends. Default implementation does nothing.
start_keyword(keyword)
Called when keyword starts. Default implementation does nothing.
Can return explicit False to stop visiting.
start_message(msg)
Called when message starts. Default implementation does nothing.
Can return explicit False to stop visiting.
start_test(test)
Called when test starts. Default implementation does nothing.
Can return explicit False to stop visiting.
visit_message(msg)
Implements visiting the message.
Can be overridden to allow modifying the passed in msg without calling start_message() or end_message().
visit_suite(suite)
Implements traversing through the suite and its direct children.
Can be overridden to allow modifying the passed in suite without calling start_suite() or end_suite() nor visiting child suites, tests or keywords (setup and teardown) at all.
robot.conf.gatherfailed.gather_failed_tests\(output\)
robot.conf.gatherfailed.gather_failed_suites\(output\)

**robot.conf.settings module**

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

get_rebot_settings()
listeners
debug_file
suite_config
randomize_seed
randomize_suites
randomize_tests
dry_run
exit_on_failure
exit_on_error
skip_teardown_on_exit
console_output_config
console_type
console_width
console_markers
max_error_lines
pre_run_modifiers
run_empty_suite
variables
variable_files
extension
console_colors
critical_tags
flatten_keywords
log
log_level
non_critical_tags
output
output_directory
pre_rebot_modifiers
remove_keywords
report
rpa
split_log
statistics_config
status_rc
xunit
xunit_skip_noncritical
class robot.conf.settings.RebotSettings (options=None, **extra_options)
   Bases: robot.conf.settings._BaseSettings
   suite_config
   log_config
   report_config
   merge
   console_output_config
   console_colors
   critical_tags
   flatten_keywords
   log
   log_level
   non_critical_tags
   output
   output_directory
   pre_rebot_modifiers
   process_empty_suite
   remove_keywords
   report
   rpa
   split_log
   statistics_config
   status_rc
   xunit
   xunit_skip_noncritical
   expand_keywords
The robot.htmldata package

This package is used for writing output files in HTML format. It is considered stable but it is not part of the public API.

Submodules

The robot.htmldata.htmlfilewriter module

The robot.htmldata.htmlfilewriter module contains the following classes:

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

- **ModelWriter**
  
  ```python
  class robot.htmldata.htmlfilewriter.ModelWriter
  Bases: robot.htmldata.htmlfilewriter._Writer
  
  handles(line)
  
  write(line)
  ```

- **LineWriter**
  
  ```python
  class robot.htmldata.htmlfilewriter.LineWriter(output)
  Bases: robot.htmldata.htmlfilewriter._Writer
  
  handles(line)
  
  write(line)
  ```

- **GeneratorWriter**
  
  ```python
  class robot.htmldata.htmlfilewriter.GeneratorWriter(html_writer)
  Bases: robot.htmldata.htmlfilewriter._Writer
  
  write(line)
  
  handles(line)
  ```

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

- **CssFileWriter**
  
  ```python
  class robot.htmldata.htmlfilewriter.CssFileWriter(html_writer, base_dir)
  Bases: robot.htmldata.htmlfilewriter._InliningWriter
  
  write(line)
  
  handles(line)
  ```

The robot.htmldata.jartemplate module

The robot.htmldata.jsonwriter module

The robot.htmldata.jsonwriter module contains the following class:

- **JsonWriter**
  
  ```python
  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.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.

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

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

**Submodules**

**robot.libdocpkg.builder module**

```python
robot.libdocpkg.builder.JavaDocBuilder()
robot.libdocpkg.builder.DocumentationBuilder(library_or_resource)
```

**robot.libdocpkg.consoleviewer module**

```python
class robot.libdocpkg.consoleviewer.ConsoleViewer(libdoc)
    Bases: object
    
    @classmethod
    handle(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**

```python
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)
    Bases: object
    
    doc_format='ROBOT')
```
html (doc, intro=False)

```python
class robot.libdocpkg.htmlwriter.DocToHtml (doc_format)
    Bases: object
```

**robot.libdocpkg.java9builder module**

**robot.libdocpkg.javabuilder module**

```python
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**

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

doc_format
keywords
all_tags
save (output=None, format='HTML')
```

```python
class robot.libdocpkg.model.KeywordDoc (name=", args=(), doc=", tags=())
    Bases: robot.utils.sortable.Sortable
    shortdoc
```

**robot.libdocpkg.output module**

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

**robot.libdocpkg.robotbuilder module**

```python
class robot.libdocpkg.robotbuilder.LibraryDocBuilder
    Bases: object

    build (library)
```

```python
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 (force_html_doc=False)
    Bases: object

    write (libdoc, outfile)

class robot.libdocpkg.xmlwriter.DocFormatter (doc_format, force_html=False)
    Bases: object

robot.libraries package

Package hosting Robot Framework standard test libraries.

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

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

Submodules

robot.libraries.BuiltIn module

robot.libraries.BuiltIn.run_keyword_variant (resolve)

class robot.libraries.BuiltIn.BuiltIn

An always available standard library with often needed keywords.

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

== Table of contents ==

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

= HTML error messages =

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

= Evaluating expressions =

Many keywords, such as Evaluate, Run Keyword If and Should Be True, accept an expression that is evaluated in Python.

== Evaluation namespace ==

Expressions are evaluated using Python’s [http://docs.python.org/library/functions.html#eval|eval] function so that all Python built-ins like `len()` and `int()` are available. In addition to that, all unrecognized variables are considered to be modules that are automatically imported. It is possible to use all available Python modules, including the standard modules and the installed third party modules.

`Evaluate` also allows configuring the execution namespace with a custom namespace and with custom modules to be imported. The latter functionality is useful when using nested modules like rootmod.submod that are implemented so that the root module does not automatically import sub modules. Otherwise the automatic module import mechanism described earlier is enough to get the needed modules imported.

NOTE: Automatic module import is a new feature in Robot Framework 3.2. Earlier modules needed to be explicitly taken into use when using the Evaluate keyword and other keywords only had access to `sys` and `os` modules.

== Using variables ==

When a variable is used in the expressing using the normal `${variable}` syntax, its value is replaced 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.

Actual variables values are also 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.

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

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

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

True examples:
False examples:

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

= Pattern matching =

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

== Glob patterns ==

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

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

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

== Regular expressions ==

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

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

= Multiline string comparison =

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

Results in the following error message:

= String representations =

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

- It is not possible to see difference between different objects that have same string representation like string 42 and integer 42. Should Be Equal and some other keywords add the type information to the error message in these cases, though.
- Non-printable characters such as the null byte are not visible.
- Trailing whitespace is not visible.
- Different newlines (\r\n on Windows, \n elsewhere) cannot be separated from each others.
• There are several Unicode characters that are different but look the same. One example is the Latin "a" and the Cyrillic "а". Error messages like "a != а" are not very helpful.

• Some Unicode characters can be represented using different forms. For example, "ä" can be represented either as a single code point \u00e4 or using two code points \u0061 and \u0308 combined together. Such forms are considered canonically equivalent, but strings containing them are not considered equal when compared in Python. Error messages like "ä != a" are not that helpful either.

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

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

The formatter argument is new in Robot Framework 3.1.2.

== str ==

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

== repr ==

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

There are also some enhancements compared to the standard repr(): - Bigger lists, dictionaries and other containers are pretty-printed so that there is one item per row.

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

== ascii ==

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

• On Python 3 non-ASCII characters are escaped like \xe4 instead of showing them as-is like ä. This makes it easier to see differences between Unicode characters that look the same but are not equal. This is how repr() works in Python 2.

• On Python 2 just uses the standard repr() meaning that Unicode strings get the u prefix and no b prefix is added to byte strings.

• Containers are not pretty-printed.
**catenate (**items**)**
Catenates the given items together and returns the resulted string.

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

**comment (**messages**)**
Displays the given messages in the log file as keyword arguments.

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

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

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

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

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

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

**convert_to_binary (item, base=None, prefix=None, length=None)**
Converts the given item to a binary string.

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

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

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

**convert_to_boolean (item)**
Converts the given item to Boolean true or false.

Handles strings True and False (case-insensitive) as expected, otherwise returns item’s [http://docs.python.org/library/stdtypes.html#truth](http://docs.python.org/library/stdtypes.html#truth) value using Python’s `bool()` method.

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

Valid input types are listed below:

- **text**: Converts text to bytes character by character. All characters with ordinal below 256 can be used and are converted to bytes with same values. Many characters are easiest to represent using escapes like \x00 or \xff. Supports both Unicode strings and bytes.
- **int**: Converts integers separated by spaces to bytes. Similarly as with *Convert To Integer*, it is possible to use binary, octal, or hex values by prefixing the values with 0b, 0o, or 0x, respectively.
- **hex**: Converts hexadecimal values to bytes. Single byte is always two characters long (e.g. 01 or FF). Spaces are ignored and can be used freely as a visual separator.
- **bin**: Converts binary values to bytes. Single byte is always eight characters long (e.g. 00001010). Spaces are ignored and can be used freely as a visual separator.
In addition to giving the input as a string, it is possible to use lists or other iterables containing individual characters or numbers. In that case numbers do not need to be padded to certain length and they cannot contain extra spaces.

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

convert_to_hex (item, base=None, prefix=None, length=None, lowercase=False)
Converts the given item to a hexadecimal string.

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

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

By default the value is returned as an upper case string, but the lowercase argument a true value (see Boolean arguments) turns the value (but not the given prefix) to lower case.

See also Convert To Integer, Convert To Binary and Convert To Octal.

convert_to_integer (item, base=None)
Converts the given item to an integer number.

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

• Give base explicitly to the keyword as base argument.

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

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

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

convert_to_number (item, precision=None)
Converts the given item to a floating point number.

If the optional precision is positive or zero, the returned number is rounded to that number of decimal digits. Negative precision means that the number is rounded to the closest multiple of 10 to the power of the absolute precision. If a number is equally close to a certain precision, it is always rounded away from zero.

Notice that machines generally cannot store floating point numbers accurately. This may cause surprises with these numbers in general and also when they are rounded. For more information see, for example, these resources:

• http://docs.python.org/tutorial/floatinptpoint.html

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

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

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

convert_to_octal (item, base=None, prefix=None, length=None)
Converts the given item to an octal string.

The item, with an optional base, is first converted to an integer using Convert To Integer internally. After that it is converted to an octal number (base 8) represented as a string such as 775.
The returned value can contain an optional prefix and can be required to be of minimum length (excluding the prefix and a possible minus sign). If the value is initially shorter than the required length, it is padded with zeros.

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

**convert_to_string(item)**

Converts the given item to a Unicode string.

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

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

**create_dictionary(*items)**

Creates and returns a dictionary based on the given items.

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

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

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

**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 result.

expression is evaluated in Python as explained in the Evaluating expressions section.

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.

Starting from Robot Framework 3.2, modules used in the expression are imported automatically. modules argument is still needed with nested modules like rootmod.submod that are implemented so that the root module does not automatically import sub modules. This is illustrated by the selenium.webdriver example below.

Variables used like ${variable} are replaced in the expression before evaluation. Variables are also available in the evaluation namespace and can be accessed using the special $variable syntax as explained in the Evaluating expressions section.

**NOTE:** Prior to Robot Framework 3.2 using modules=rootmod.submod was not enough to make the root module itself available in the evaluation namespace. It needed to be taken into use explicitly like modules=rootmod, rootmod.submod.

**exit_for_loop()**

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

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

**exit_for_loop_if** *(condition)*

Stops executing the enclosing for loop if the *condition* is true.

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

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

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

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

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

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

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

Stops the whole test execution.

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

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

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

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

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

**get_length** *(item)*

Returns and logs the length of the given item as an integer.

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

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

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

Returns the currently active instance of the specified test library.

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

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

If the optional argument all is given a true value, then a dictionary mapping all library names to instances will be returned.

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

Returns the given time in the requested format.

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

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

1) If format contains the word *epoch*, the time is returned in seconds after the UNIX epoch (1970-01-01 00:00:00 UTC). The return value is always an integer.

2) If format contains any of the words *year*, *month*, *day*, *hour*, *min*, or *sec*, only the selected parts are returned. The order of the returned parts is always the one in the previous sentence and the order of words in format is not significant. The parts are returned as zero-padded strings (e.g. May -> 05).

3) Otherwise (and by default) the time is returned as a timestamp string in the format 2006-02-24 15:08:31.

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

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

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

3) If time is equal to NOW (default), the current local time is used.

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

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

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

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

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

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

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

**get_variables** *(no_decoration=False)*

Returns a dictionary containing all variables in the current scope.

Variables are returned as a special dictionary that allows accessing variables in space, case, and underscore insensitive manner similarly as accessing variables in the test data. This dictionary supports all same operations as normal Python dictionaries and, for example, Collections library can be used to access or modify it. Modifying the returned dictionary has no effect on the variables available in the current scope.

By default variables are returned with ${}, @{} or &{} decoration based on variable types. Giving a true value (see Boolean arguments) to the optional argument no-decoration will return the variables without the decoration.
import_library (name, *args)
Imports a library with the given name and optional arguments.

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

This keyword supports importing libraries both using library names and physical paths. When paths are used, they must be given in absolute format or found from [http://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#pythonpath-jythonpath-and-ironpythonpath| search path]. Forward slashes can be used as path separators in all operating systems.

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

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

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

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

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

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

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

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

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

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

See also Variable Should Exist.

length_should_be (item, length, msg=None)
Verifies that the length of the given item is correct.

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

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

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

Messages logged with the WARN or ERROR levels will be automatically visible also in the console and in the Test Execution Errors section in the log file.
If the `html` argument is given a true value (see Boolean arguments), the message will be considered HTML and special characters such as `<` are not escaped. For example, logging `<img src="image.png">` creates an image when `html` is true, but otherwise the message is that exact string. An alternative to using the `html` argument is using the HTML pseudo log level. It logs the message as HTML using the INFO level.

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

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

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

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

```
log_many(*messages)
```

Logs the given messages as separate entries using the INFO level.

Supports also logging list and dictionary variable items individually.

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

```
log_to_console(message, stream='STDOUT', no_newline=False)
```

Logs the given message to the console.

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

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

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

```
log_variables(level='INFO')
```

Logs all variables in the current scope with given log level.

```
no_operation()
```

Does absolutely nothing.

```
pass_execution(message, *tags)
```

Skips rest of the current test, setup, or teardown with PASS status.

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

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

- When used in a test outside setup or teardown, passes that particular test case. Possible test and keyword teardowns are executed.

Possible continuable failures before this keyword is used, as well as failures in executed teardowns, will fail the execution.
It is mandatory to give a message explaining why execution was passed. By default the message is considered plain text, but starting it with `*HTML*` allows using HTML formatting.

It is also possible to modify test tags passing tags after the message similarly as with `Fail` keyword. Tags starting with a hyphen (e.g. `-regression`) are removed and others added. Tags are modified using `Set Tags` and `Remove Tags` internally, and the semantics setting and removing them are the same as with these keywords.

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

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

```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`.

```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.

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

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

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

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

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

**return_from_keyword_if (condition, **return_values**)**

Returns from the enclosing user keyword if `condition` is true.

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

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

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

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

Executes the given keyword with the given arguments.

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

**run_keyword_and_continue_on_failure (name, **args**)**

Runs the keyword and continues execution even if a failure occurs.

The keyword name and arguments work as with `Run Keyword`.

The execution is not continued if the failure is caused by invalid syntax, timeout, or fatal exception.

**run_keyword_and_expect_error (expected_error, name, **args**)**

Runs the keyword and checks that the expected error occurred.
The keyword to execute and its arguments are specified using name and *args exactly like with Run Keyword.

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

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

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

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

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.

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

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

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

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

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

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

run_keyword_and_return_status (name, *args)
Runs the given keyword with given arguments and returns the status as a Boolean value.

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

The keyword name and arguments work as in Run Keyword.

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

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

The given condition is evaluated in Python as explained in Evaluating expressions, and name and *args have same semantics as with Run Keyword.
In this example, only either *Some Action* or *Another Action* is executed, based on the status of *My Keyword*. Instead of *Run Keyword And Ignore Error* you can also use *Run Keyword And Return Status*.

Variables used like `${variable}`, as in the examples above, are replaced in the expression before evaluation. Variables are also available in the evaluation namespace and can be accessed using special syntax `$variable` as explained in the *Evaluating expressions* section.

This keyword supports also optional ELSE and ELSE IF branches. Both of them are defined in `*args` and must use exactly format `ELSE` or `ELSE IF`, respectively. ELSE branches must contain first the name of the keyword to execute and then its possible arguments. ELSE IF branches must first contain a condition, like the first argument to this keyword, and then the keyword to execute and its possible arguments. It is possible to have ELSE branch after ELSE IF and to have multiple ELSE IF branches. Nested *Run Keyword If* usage is not supported when using ELSE and/or ELSE IF branches.

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

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

In this example, `${var2}` will be set to `None` if `${condition}` is false. Notice that `ELSE` and `ELSE IF` control words must be used explicitly and thus cannot come from variables. If you need to use literal `ELSE` and `ELSE IF` strings as arguments, you can escape them with a backslash like `\ELSE` and `\ELSE IF`.

Python’s [http://docs.python.org/library/os.html#os] and [http://docs.python.org/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.

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.

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.

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.

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.

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

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

run_keyword_unless(condition, name, *args)
  Runs the given keyword with the given arguments if condition is false.
  See Run Keyword If for more information and an example. Notice that this keyword does not support ELSE or ELSE IF branches like Run Keyword If does, though.

run_keywords(*keywords)
  Executes all the given keywords in a sequence.
  This keyword is mainly useful in setups and teardowns when they need to take care of multiple actions and creating a new higher level user keyword would be an overkill.
  By default all arguments are expected to be keywords to be executed.
  Keywords can also be run with arguments using upper case AND as a separator between keywords. The keywords are executed so that the first argument is the first keyword and proceeding arguments until the first AND are arguments to it. First argument after the first AND is the second keyword and proceeding arguments until the next AND are its arguments. And so on.
  Notice that the AND control argument must be used explicitly and cannot itself come from a variable. If you need to use literal AND string as argument, you can either use variables or escape it with a backslash like \AND.

set_global_variable(name, *values)
  Makes a variable available globally in all tests and suites.
  Variables set with this keyword are globally available in all subsequent test suites, test cases and user keywords. Also variables in variable tables are overridden. Variables assigned locally based on keyword return values or by using Set Test Variable and Set Suite Variable override these variables in that scope, but the global value is not changed in those cases.
  In practice setting variables with this keyword has the same effect as using command line options --variable and --variablefile. Because this keyword can change variables everywhere, it should be used with care.
  See Set Suite Variable for more information and examples.

set_library_search_order(*search_order)
  Sets the resolution order to use when a name matches multiple keywords.
  The library search order is used to resolve conflicts when a keyword name in the test data matches multiple keywords. The first library (or resource, see below) containing the keyword is selected and that keyword implementation used. If the keyword is not found from any library (or resource), test executing fails the same way as when the search order is not set.
  When this keyword is used, there is no need to use the long LibraryName.Keyword Name notation. For example, instead of having
  you can have
This keyword can be used also to set the order of keywords in different resource files. In this case resource names must be given without paths or extensions like:

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

- The old order is returned and can be used to reset the search order later.
- Library and resource names in the search order are both case and space insensitive.

**set_local_variable** *(name, *values)*

Makes a variable available everywhere within the local scope.

Variables set with this keyword are available within the local scope of the currently executed test case or in the local scope of the keyword in which they are defined. For example, if you set a variable in a user keyword, it is available only in that keyword. Other test cases or keywords will not see variables set with this keyword.

This keyword is equivalent to a normal variable assignment based on a keyword return value.

is equivalent with

This keyword will provide the option of setting local variables inside keywords like **Run Keyword If**, **Run Keyword And Return If**, **Run Keyword Unless** which until now was not possible by using **Set Variable**.

It will also be possible to use this keyword from external libraries that want to set local variables.

New in Robot Framework 3.2.

**set_log_level** *(level)*

Sets the log threshold to the specified level and returns the old level.

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

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

**set_suite_documentation** *(doc, append=False, top=False)*

Sets documentation for the current test suite.

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

This keyword sets the documentation of the current suite by default. If the optional **top** argument is given a true value (see **Boolean arguments**), the documentation of the top level suite is altered instead.

The documentation of the current suite is available as a built-in variable **${SUITE DOCUMENTATION}**.

**set_suite_metadata** *(name, value, append=False, top=False)*

Sets metadata for the current test suite.

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

This keyword sets the metadata of the current suite by default. If the optional **top** argument is given a true value (see **Boolean arguments**), the metadata of the top level suite is altered instead.

The metadata of the current suite is available as a built-in variable **${SUITE METADATA}** in a Python dictionary. Notice that modifying this variable directly has no effect on the actual metadata the suite has.
**set_suite_variable** (name, *values)

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

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

Possible child test suites do not see variables set with this keyword by default, but that can be controlled by using `children=<option>` as the last argument. If the specified `<option>` given a true value (see **Boolean arguments**), the variable is set also to the child suites. Parent and sibling suites will never see variables set with this keyword.

The name of the variable can be given either as a normal variable name (e.g. `${NAME}`) or in escaped format as \${NAME} or $NAME. Variable value can be given using the same syntax as when variables are created in the Variable table.

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

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

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

This limitation applies also to **Set Test Variable**, **Set Global Variable**, **Variable Should Exist**, **Variable Should Not Exist** and **Get Variable Value** keywords.

**set_tags** (*tags*)

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

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

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

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

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

**set_task_variable** (name, *values)

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

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

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

Sets documentation for the current test case.

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

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

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

Sets message for the current test case.

If the optional `append` argument is given a true value (see **Boolean arguments**), the given `message` is added after the possible earlier message by joining the messages with a space.
In test teardown this keyword can alter the possible failure message, but otherwise failures override messages set by this keyword. Notice that in teardown the message is available as a built-in variable `${TEST MESSAGE}`.

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

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

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

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

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

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

**set_variable**(values)**

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

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

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

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

Sets variable based on the given condition.

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

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

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

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

Verifies that the given item is empty.

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

**should_be_equal**(first, second, msg=None, values=True, ignore_case=False, formatter='str'**

Fails if the given objects are unequal.

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

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

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

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

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

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

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

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

The conversion is done with *Convert To Number* keyword using the given `precision`.

As discussed in the documentation of *Convert To Number*, machines generally cannot store floating point numbers accurately. Because of this limitation, comparing floats for equality is problematic and a correct approach to use depends on the context. This keyword uses a very naive approach of rounding the numbers before comparing them, which is both prone to rounding errors and does not work very well if numbers are really big or small. For more information about comparing floats, and ideas on how to implement your own context specific comparison algorithm, see [http://randomascii.wordpress.com/2012/02/25/comparing-floating-point-numbers-2012-edition/](http://randomascii.wordpress.com/2012/02/25/comparing-floating-point-numbers-2012-edition/).

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

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

`should_be_equal_as_strings(first, second, msg=None, values=True, ignore_case=False, for-matter='str')`
Fails if objects are unequal after converting them to strings.

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

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

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

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

`should_be_true(condition, msg=None)`
Fails if the given condition is not true.

If `condition` is a string (e.g. `${rc} < 10`), it is evaluated as a Python expression as explained in *Evaluating expressions* and the keyword status is decided based on the result. If a non-string item is given, the status is got directly from its [http://docs.python.org/library/stdtypes.html#truthtruth 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 `$variable` syntax as explained in the Evaluating expressions section.

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

`should_contain`(`container`, `item`, `msg=None`, `values=True`, `ignore_case=False`)  
Fails if `container` does not contain `item` one or more times.  
Works with strings, lists, and anything that supports Python’s `in` operator.  
See `Should Be Equal` for an explanation on how to override the default error message with arguments `msg` and `values`.  
If `ignore_case` is given a true value (see Boolean arguments) and compared items are strings, it indicates that comparison should be case-insensitive. If the `container` is a list-like object, string items in it are compared case-insensitively. New option in Robot Framework 3.0.1.

`should_contain_any`(`container`, `*items`, `**configuration`)  
Fails if `container` does not contain any of the `*items`.  
Works with strings, lists, and anything that supports Python’s `in` operator.  
Supports additional configuration parameters `msg`, `values` and `ignore_case`, which have exactly the same semantics as arguments with same names have with `Should Contain`. These arguments must always be given using `name=value` syntax after all `items`.  
Note that possible equal signs in `items` must be escaped with a backslash (e.g. `foo\=bar`) to avoid them to be passed in as `**configuration`.  
New in Robot Framework 3.0.1.

`should_contain_x_times`(`item1`, `item2`, `count`, `msg=None`, `ignore_case=False`)  
Fails if `item1` does not contain `item2` `count` times.  
Works with strings, lists and all objects that `Get Count` works with. The default error message can be overridden with `msg` and the actual count is always logged.  
If `ignore_case` is given a true value (see Boolean arguments) and compared items are strings, it indicates that comparison should be case-insensitive. If the `item1` is a list-like object, string items in it are compared case-insensitively. New option in Robot Framework 3.0.1.

`should_end_with`(`str1`, `str2`, `msg=None`, `values=True`, `ignore_case=False`)  
Fails if the string `str1` does not end with the string `str2`.  
See `Should Be Equal` for an explanation on how to override the default error message with `msg` and `values`, as well as for semantics of the `ignore_case` option.

`should_match`(`string`, `pattern`, `msg=None`, `values=True`, `ignore_case=False`)  
Fails if the given `string` does not match the given `pattern`.  
Pattern matching is similar as matching files in a shell with `*`, `?` and `[chars]` acting as wildcards. See the Glob patterns section for more information.  
See `Should Be Equal` for an explanation on how to override the default error message with `msg` and `values`, as well as for semantics of the `ignore_case` option.

`should_match_regexp`(`string`, `pattern`, `msg=None`, `values=True`)  
Fails if `string` does not match `pattern` as a regular expression.  
See the Regular expressions section for more information about regular expressions and how to use them in Robot Framework test data.
Notice that the given pattern does not need to match the whole string. For example, the pattern `ello` matches the string `Hello world!`. If a full match is needed, the `^` and `$` characters can be used to denote the beginning and end of the string, respectively. For example, `^ello$` only matches the exact string `ello`.

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

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

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

**should_not_be_empty**(item, `msg=None`)
Verifies that the given item is not empty.

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

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

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

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

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

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

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

See `Should Be Equal As Integers` for some usage examples.

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

The conversion is done with `Convert To Number` keyword using the given `precision`.

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

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

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

If `ignore_case` is given a true value (see `Boolean arguments`), comparison is done case-insensitively. Strings are always [NFC normalized].

`ignore_case` is a new feature in Robot Framework 3.0.1.
**should_not_be_true** *(condition, msg=None)*
Fails if the given condition is true.

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

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

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

See *Should Be Equal* for an explanation on how to override the default error message with arguments *msg* and *values*. *ignore_case* has exactly the same semantics as with *Should Contain*. These arguments must always be given using `name=value` syntax after all *items*.

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

New in Robot Framework 3.0.1.

**should_not_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*.

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

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

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

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

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

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

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

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

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

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

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

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

**time** may be either a number or a time string. Time strings are in a format such as 1 day 2 hours 3 minutes 4 seconds 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.

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

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

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

See also Variable Should Not Exist and Keyword Should Exist.

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

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

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

See also Variable Should Exist and Keyword Should Exist.

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

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

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

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

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

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

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

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

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

**args**

**message**

robot.libraries.BuiltIn.register_run_keyword (library, keyword, args_to_process=None, deprecation_warning=True)

Registers ‘run keyword’ so that its arguments can be handled correctly.

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

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

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

Keywords executed by registered run keywords can be tested in dry-run mode if they have ‘name’ argument which takes the name of the executed keyword.

2) How to use this method

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

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

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

3) Examples

```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`

```python
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 an empty string or equal to FALSE, NONE, NO, OFF or 0, case-insensitively. Keywords verifying something that allow dropping actual and expected values from the possible error message also consider string no values to be false. Other strings are considered true regardless their value, and other argument types are tested using the same [http://docs.python.org/library/stdtypes.html#truth|rules as in Python].

True examples:

False examples:

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

= Data in examples =

List related keywords use variables in format ${Lx}$ in their examples. They mean lists with as many alphabetic characters as specified by x. For example, ${L1}$ means ['a'] and ${L3}$ means ['a', 'b', 'c'].

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

ROBOT_LIBRARY_SCOPE = 'GLOBAL'

ROBOT_LIBRARY_VERSION = '3.2b3.dev1'

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

Fails if pattern is not found in list.

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

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

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

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

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

Use the msg argument to override the default error message.

See also Should Not Contain Match.

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

Fails if pattern is found in list.

Exact opposite of Should Contain Match keyword. See that keyword for information about arguments and usage in general.
get_matches (list, pattern, case_insensitive=False, whitespace_insensitive=False)
Returns a list of matches to pattern in list.

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

get_match_count (list, pattern, case_insensitive=False, whitespace_insensitive=False)
Returns the count of matches to pattern in list.

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

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

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

The given lists are not altered by this keyword.

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

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

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

New in Robot Framework 2.9.

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

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

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

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

The given dictionary is never altered by this keyword.

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

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

The given list is never altered by this keyword.

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

The search can be narrowed to the selected sublist by the start and end indexes having the same semantics as with Get Slice From List keyword. The given list is never altered by this keyword.

dictionaries_should_be_equal (dict1, dict2, msg=None, values=True)
Fails if the given dictionaries are not equal.
First the equality of dictionaries’ keys is checked and after that all the key value pairs. If there are differences between the values, those are listed in the error message. The types of the dictionaries do not need to be same.

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

\[\text{dictionary\_should\_contain\_item}(\text{dictionary}, \text{key}, \text{value}, \text{msg}=\text{None})\]

An item of key/value must be found in a dictionary.

Value is converted to unicode for comparison.

Use the \text{msg} argument to override the default error message.

\[\text{dictionary\_should\_contain\_key}(\text{dictionary}, \text{key}, \text{msg}=\text{None})\]

Fails if key is not found from dictionary.

Use the \text{msg} argument to override the default error message.

\[\text{dictionary\_should\_contain\_sub\_dictionary}(\text{dict1}, \text{dict2}, \text{msg}=\text{None}, \text{values}=\text{True})\]

Fails unless all items in \text{dict2} are found from \text{dict1}.

See Lists Should Be Equal for more information about configuring the error message with \text{msg} and values arguments.

\[\text{dictionary\_should\_contain\_value}(\text{dictionary}, \text{value}, \text{msg}=\text{None})\]

Fails if value is not found from dictionary.

Use the \text{msg} argument to override the default error message.

\[\text{dictionary\_should\_not\_contain\_key}(\text{dictionary}, \text{key}, \text{msg}=\text{None})\]

Fails if key is found from dictionary.

Use the \text{msg} argument to override the default error message.

\[\text{dictionary\_should\_not\_contain\_value}(\text{dictionary}, \text{value}, \text{msg}=\text{None})\]

Fails if value is found from dictionary.

Use the \text{msg} argument to override the default error message.

\[\text{get\_dictionary\_items}(\text{dictionary}, \text{sort\_keys}=\text{True})\]

Returns items of the given dictionary as a list.

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

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

The given dictionary is never altered by this keyword.

\text{sort\_keys} is a new option in Robot Framework 3.1.2. Earlier items were always sorted based on keys.

\[\text{get\_dictionary\_keys}(\text{dictionary}, \text{sort\_keys}=\text{True})\]

Returns keys of the given dictionary as a list.

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

The given dictionary is never altered by this keyword.

\text{sort\_keys} is a new option in Robot Framework 3.1.2. Earlier keys were always sorted.
get_dictionary_values (dictionary, sort_keys=True)
Returns values of the given dictionary as a list.

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

The given dictionary is never altered by this keyword.

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

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

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

The given dictionary is never altered by this keyword.

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

The given list is never altered by this keyword.

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

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

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

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

The given list is never altered by this keyword.

If both start and end are given, a sublist containing values from start to end is returned. This is the same as list[start:end] in Python. To get all items from the beginning, use 0 as the start value, and to get all items until and including the end, use None (default) as the end value.

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

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

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

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

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

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

list_should_contain_sub_list (list1, list2, msg=None, values=True)
Fails if not all of the elements in list2 are found in list1.
The order of values and the number of values are not taken into account.

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

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

Use the msg argument to override the default error message.

**list_should_not_contain_duplicates** *(list_, msg=None)*  
Fails if any element in the list is found from it more than once.

The default error message lists all the elements that were found from the list multiple times, but it can be overridden by giving a custom msg. All multiple times found items and their counts are also logged.

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

**list_should_not_contain_value** *(list_, value, msg=None)*  
Fails if the value is found from list.

Use the msg argument to override the default error message.

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

The keyword first verifies that the lists have equal lengths, and then it checks are all their values equal. Possible differences between the values are listed in the default error message like Index 4: ABC != Abc. The types of the lists do not need to be the same. For example, Python tuple and list with same content are considered equal.

The error message can be configured using msg and values arguments: - If msg is not given, the default error message is used. - If msg is given and values gets a value considered true (see Boolean arguments), the error message starts with the given msg followed by a newline and the default message.

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

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

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

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

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

If you only want to log the size, use keyword Get Length from the BuiltIn library.

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

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

If you only want to the length, use keyword Get Length from the BuiltIn library.

**pop_from_dictionary** *(dictionary, key, default=)*  
Pops the given key from the dictionary and returns its value.
By default the keyword fails if the given key cannot be found from the dictionary. If optional default value is given, it will be returned instead of failing.

New in Robot Framework 2.9.2.

`remove_duplicates(list_)
Returns a list without duplicates based on the given list.

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

`remove_from_dictionary(dictionary, *keys)
Removes the given keys from the dictionary.

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

`remove_from_list(list_, index)
Removes and returns the value specified with an index from list.

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

`remove_values_from_list(list_, *values)
Removes all occurrences of given values from list.

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

`reverse_list(list_)
Reverses the given list in place.

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

`set_list_value(list_, index, value)
Sets the value of list specified by index to the given value.

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

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

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

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

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

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

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

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


### robot.libraries.DateTime module

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

= Table of Contents =
  • Terminology
  • Date formats
  • Time formats
  • Millisecond handling
  • Programmatic usage
  • Shortcuts
  • Keywords

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

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


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

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

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

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

== Custom timestamp ==
It is possible to use custom timestamps in both input and output. The custom format is same as accepted by Python’s [http://docs.python.org/library/datetime.html#strftime-strptime-behavior|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

== Python datetime ==

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

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

== Epoch time ==

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

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

== Earliest supported date ==

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

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

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

= Time formats =

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

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

== Number ==

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

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

== Time string ==

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

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

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

== Timer string ==

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

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

== Python timedelta ==

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

= Millisecond handling =

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

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

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

= Programmatic usage =

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

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

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

Returns current local or UTC time with an optional increment.
Arguments: - time_zone: Get the current time on this time zone. Currently only
  local (default) and UTC are supported.
  - increment: Optional time increment to add to the returned date in one of the supported time for-
    mats. 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 milli-
    second handling.

\[\text{robot.libraries.DateTime.convert_date}(\text{date}, \text{result_format='timestamp'}, \text{ex-
  clude_millis=False, date_format=None})\]

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.

\[\text{robot.libraries.DateTime.convert_time}(\text{time}, \text{result_format='number'}, \text{ex-
  clude_millis=False})\]

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.

\[\text{robot.libraries.DateTime.subtract_date_from_date}(\text{date1}, \text{date2}, \text{re-
  sult_format='number', exclude_millis=False, date1_format=None, date2_format=None})\]

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 milli-
    second handling.

  - date1_format: Possible custom timestamp format of date1.

  - date2_format: Possible custom timestamp format of date2.

Examples:

\[\text{robot.libraries.DateTime.add_time_to_date}(\text{date}, \text{time}, \text{result_format='timestamp'}, \text{ex-
  clude_millis=False, date_format=None})\]

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*.
- **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. If you want to wrap lines manually, you can add newlines using the \n character sequence.

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

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

Pauses test execution until user clicks Ok button.

message is the message shown in the dialog.

robot.libraries.Dials.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.Dials.getvalue_from_user(message, default_value=' ', hidden=False)

Pauses test execution and asks user to input a value.

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

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

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

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

robot.libraries.Dials.getselection_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.Dials.getselections_from_user(message, *values)

Pauses test execution and asks user to select multiple values.

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

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

New in Robot Framework 3.1.

robot.libraries.Easter module

robot.libraries.Easter.none_shall_pass(who)
robot.libraries.OperatingSystem module

class robot.libraries.OperatingSystem.OperatingSystem

    Bases: object

    A test library providing keywords for OS related tasks.

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

== Table of contents ==

• Path separators
• Pattern matching
• Tilde expansion
• Boolean arguments
• Example
• Shortcuts
• Keywords

= Path separators =

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

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

= Pattern matching =

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

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

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

= Tilde expansion =

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

The ~username form does not work on Jython.

= Boolean arguments =

Some keywords accept arguments that are handled as Boolean values true or false. If such an argument is
given as a string, it is considered false if it is an empty string or equal to FALSE, NONE, NO, OFF or 0, case-
insensitively. Other strings are considered true regardless their value, and other argument types are tested using
the same [http://docs.python.org/library/stdtypes.html#truthrules as in Python].
True examples:
False examples:
Considering string NONE false is new in Robot Framework 3.0.3 and considering also OFF and 0 false is new in Robot Framework 3.1.

= Example =

ROBOT_LIBRARY_SCOPE = 'GLOBAL'
ROBOT_LIBRARY_VERSION = '3.2b3.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 [http://robotframework.org/robotframework/latest/libraries/Process.html| Process library] supports better process configuration and is generally recommended as a replacement for this keyword.

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

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

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

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

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

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

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

get_file (path, encoding='UTF-8', encoding_errors='strict')
Returns the contents of a specified file.

This keyword reads the specified file and returns the contents. Line breaks in content are converted to platform independent form. See also Get Binary File.
encoding defines the encoding of the file. The default value is UTF-8, which means that UTF-8 and ASCII encoded files are read correctly. In addition to the encodings supported by the underlying Python implementation, the following special encoding values can be used:

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

echo encoding_errors argument controls what to do if decoding some bytes fails. All values accepted by decode method in Python are valid, but in practice the following values are most useful:

- strict: Fail if characters cannot be decoded (default).
- ignore: Ignore characters that cannot be decoded.
- replace: Replace characters that cannot be decoded with a replacement character.

Support for SYSTEM and CONSOLE encodings in Robot Framework 3.0.

get_binary_file(path)
Returns the contents of a specified file.

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

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

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

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

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

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

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

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

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

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

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

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

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

The path can be given as an exact path or as a glob pattern. The pattern matching syntax is explained in introduction. The default error message can be overridden with the msg argument.
file_should_not_exist (path, msg=None)
Fails if the given path points to an existing file.

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

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

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

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

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

wait_until_removed (path, timeout='1 minute')
Waits until the given file or directory is removed.

The path can be given as an exact path or as a glob pattern. The pattern matching syntax is explained in introduction. If the path is a pattern, the keyword waits until all matching items are removed.

The optional timeout can be used to control the maximum time of waiting. The timeout is given as a timeout string, e.g. in a format 15 seconds, 1min 10s or just 10. The time string format is described in an appendix of Robot Framework User Guide.

If the timeout is negative, the keyword is never timed-out. The keyword returns immediately, if the path does not exist in the first place.

wait_until_created (path, timeout='1 minute')
Waits until the given file or directory is created.

The path can be given as an exact path or as a glob pattern. The pattern matching syntax is explained in introduction. If the path is a pattern, the keyword returns when an item matching it is created.

The optional timeout can be used to control the maximum time of waiting. The timeout is given as a timeout string, e.g. in a format 15 seconds, 1min 10s or just 10. The time string format is described in an appendix of Robot Framework User Guide.

If the timeout is negative, the keyword is never timed-out. The keyword returns immediately, if the path already exists.

directory_should_be_empty (path, msg=None)
Fails unless the specified directory is empty.

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

directory_should_not_be_empty (path, msg=None)
Fails if the specified directory is empty.

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

file_should_be_empty (path, msg=None)
Fails unless the specified file is empty.

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

file_should_not_be_empty (path, msg=None)
Fails if the specified directory is empty.

The default error message can be overridden with the msg argument.
**create_file** *(path, content='', encoding='UTF-8')*

Creates a file with the given content and encoding.

- If the directory where the file is created does not exist, it is automatically created along with possible missing intermediate directories. Possible existing file is overwritten.
- On Windows newline characters (`\n`) in content are automatically converted to Windows native newline sequence (`\r\n`).
- See **Get File** for more information about possible **encoding** values, including special values **SYSTEM** and **CONSOLE**.
- Use **Append To File** if you want to append to an existing file and **Create Binary File** if you need to write bytes without encoding. **File Should Not Exist** can be used to avoid overwriting existing files.

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

**create_binary_file** *(path, content)*

Creates a binary file with the given content.

- If content is given as a Unicode string, it is first converted to bytes character by character. All characters with ordinal below 256 can be used and are converted to bytes with same values. Using characters with higher ordinal is an error.
- Byte strings, and possible other types, are written to the file as is.
- If the directory for the file does not exist, it is created, along with missing intermediate directories.
- Use **Create File** if you want to create a text file using a certain encoding. **File Should Not Exist** can be used to avoid overwriting existing files.

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

Appends the given content to the specified file.

- If the file exists, the given text is written to its end. If the file does not exist, it is created.
- Other than not overwriting possible existing files, this keyword works exactly like **Create File**. See its documentation for more details about the usage.

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

**remove_file** *(path)*

Removes a file with the given path.

- Passes if the file does not exist, but fails if the path does not point to a regular file (e.g. it points to a directory).
- The path can be given as an exact path or as a glob pattern. The pattern matching syntax is explained in **introduction**. If the path is a pattern, all files matching it are removed.

**remove_files** *(paths)*

Uses **Remove File** to remove multiple files one-by-one.

**empty_directory** *(path)*

Deletes all the content from the given directory.

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

**create_directory** *(path)*

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

**remove_directory**(path, recursive=False)

Removes the directory pointed to by the given path.

If the second argument recursive is given a true value (see Boolean arguments), the directory is removed recursively. Otherwise removing fails if the directory is not empty.

If the directory pointed to by the path does not exist, the keyword passes, but it fails, if the path points to a file.

**copy_file**(source, destination)

Copies the source file into the destination.

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

1) If the destination is an existing file, the source file is copied over it.

2) If the destination is an existing directory, the source file is copied into it. A possible file with the same name as the source is overwritten.

3) If the destination does not exist and it ends with a path separator (/ or \
), it is considered a directory. That directory is created and a source file copied into it. Possible missing intermediate directories are also created.

4) If the destination does not exist and it does not end with a path separator, it is considered a file. If the path to the file does not exist, it is created.

The resulting destination path is returned since Robot Framework 2.9.2.

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

**move_file**(source, destination)

Moves the source file into the destination.

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

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

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

**copy_files**(sources_and_destination)

Copies specified files to the target directory.

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

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

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

**move_files**(sources_and_destination)

Moves specified files to the target directory.

Arguments have exactly same semantics as with Copy Files keyword.

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

**copy_directory**(source, destination)

Copies the source directory into the destination.
If the destination exists, the source is copied under it. Otherwise the destination directory and the possible missing intermediate directories are created.

**move_directory** *(source, destination)*
Moves the source directory into a destination.

Uses Copy Directory keyword internally, and source and destination arguments have exactly same semantics as with that keyword.

**get_environment_variable** *(name, default=None)*
Returns the value of an environment variable with the given name.

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

Returned variables are automatically decoded to Unicode using the system encoding.

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

**set_environment_variable** *(name, value)*
Sets an environment variable to a specified value.

Values are converted to strings automatically. Set variables are automatically encoded using the system encoding.

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

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

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

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

Does nothing if the environment variable is not set.

It is possible to remove multiple variables by passing them to this keyword as separate arguments.

**environment_variable_should_be_set** *(name, msg=None)*
Fails if the specified environment variable is not set.

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

**environment_variable_should_not_be_set** *(name, msg=None)*
Fails if the specified environment variable is set.

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

**get_environment_variables** *
Returns currently available environment variables as a dictionary.

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

**log_environment_variables** *(level='INFO')*
Logs all environment variables using the given log level.

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

**join_path** *(base, *parts)*
Joins the given path part(s) to the given base path.
The path separator (/ or \) is inserted when needed and the possible absolute paths handled as expected. The resulted path is also normalized.

- `${path}` = ‘my/path’
- `${p2}` = ‘my/path’
- `${p3}` = ‘my/path/my/file.txt’
- `${p4}` = ‘/path’
- `${p5}` = ‘/my/path2’

**join_paths** (base, *paths)

Joins given paths with base and returns resulted paths.

See *Join Path* for more information.

- `@{p1}` = [‘base/example’, ‘base/other’]
- `@{p2}` = [‘/example’, ‘/my/base/other’]
- `@{p3}` = [‘my/base/example/path’, ‘my/base/other’, ‘my/base/one/more’]

**normalize_path** (path, case_normalize=False)

Normalizes the given path.

- Collapses redundant separators and up-level references.
- Converts / to \ on Windows.
- Replaces initial ~ or ~user by that user’s home directory. The latter is not supported on Jython.
- If *case_normalize* is given a true value (see *Boolean arguments*) on Windows, converts the path to all lowercase. New in Robot Framework 3.1.

- `${path1}` = ‘abc’
- `${path2}` = ‘def’
- `${path3}` = ‘abc/def/ghi’
- `${path4}` = ‘/home/robot/stuff’

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

**split_path** (path)

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

The given path is first normalized (e.g. a possible trailing path separator is removed, special directories .. and . removed). The parts that are split are returned as separate components.

- `${path1}` = ‘abc’ & `${dir}` = ‘def’
- `${path2}` = ‘abc/def’ & `${file}` = ‘ghi.txt’
- `${path3}` = ‘def’ & `${d2}` = ‘ghi’

**split_extension** (path)

Splits the extension from the given path.

The given path is first normalized (e.g. possible trailing path separators removed, special directories .. and . removed). The base path and extension are returned as separate components so that the dot used as an extension separator is removed. If the path contains no extension, an empty string is returned for it. Possible leading and trailing dots in the file name are never considered to be extension separators.

- `${path}` = ‘file’ & `${ext}` = ‘extension’
get_modified_time (path, format='timestamp')

Returns the last modification time of a file or directory. How time is returned is determined based on the given format string as follows. Note that all checks are case-insensitive. Returned time is also automatically logged.

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

2006-03-29 15:06:21): - ${time} = '2006-03-29 15:06:21' - ${secs} = 1143637581 - ${year} = '2006' - ${y} = '2006' & ${d} = '29' - @{time} = ['2006', '03', '29', '15', '06', '21']

set_modified_time (path, mtime)

Sets the file modification and access times. Changes the modification and access times of the given file to the value determined by mtime. The time can be given in different formats described below. Note that all checks involving strings are case-insensitive. Modified time can only be set to regular files.

1) If mtime is a number, or a string that can be converted to a number, it is interpreted as seconds since the UNIX epoch (1970-01-01 00:00:00 UTC). This documentation was originally written about 1177654467 seconds after the epoch.
2) If mtime is a timestamp, that time will be used. Valid timestamp formats are YYYY-MM-DD hh:mm:ss and YYYYMMDD hhmmss.
3) If mtime is equal to NOW, the current local time is used.
4) If mtime is equal to UTC, the current time in [http://en.wikipedia.org/wiki/Coordinated_Universal_Time|UTC] is used.
5) If mtime is in the format like NOW - 1 day or UTC + 1 hour 30 min, the current local/UTC time plus/minus the time specified with the time string is used. The time string format is described in an appendix of Robot Framework User Guide.

get_file_size (path)

Returns and logs file size as an integer in bytes.

list_directory (path, pattern=None, absolute=False)

Returns and logs items in a directory, optionally filtered with pattern.

File and directory names are returned in case-sensitive alphabetical order, e.g. ['A Name', 'Second', 'a lower case name', 'one more']. Implicit directories . and .. are not returned. The returned items are automatically logged.
File and directory names are returned relative to the given path (e.g. 'file.txt') by default. If you want them be returned in absolute format (e.g. '/home/robot/file.txt'), give the absolute argument a true value (see Boolean arguments).

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

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

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

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

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

**count_files_in_directory** *(path, pattern=None)*
Wrapper for Count Items In Directory returning only file count.

**count_directories_in_directory** *(path, pattern=None)*
Wrapper for Count Items In Directory returning only directory count.

**touch** *(path)*
Emulates the UNIX touch command.

Creates a file, if it does not exist. Otherwise changes its access and modification times to the current time.

Fails if used with the directories or the parent directory of the given file does not exist.

### robot.libraries.Process module

**class** robot.libraries.Process.Process

Bases: object

Robot Framework test library for running processes.


The library has following main usages:

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

== Table of contents ==

- Specifying command and arguments
- Process configuration
- Active process
- Result object
- Boolean arguments
- Example
= Specifying command and arguments =

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

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

Possible non-string arguments are converted to strings automatically.

= Process configuration =

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

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

== Running processes in shell ==

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

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

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

== Current working directory ==

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

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

== Environment variables ==

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

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

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

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

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

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

If you are not interested in outputs at all, you can explicitly ignore them by using a special value `DEVNULL` both with `stdout` and `stderr`. For example, `stdout=DEVNULL` is the same as redirecting output on console with `>` /dev/null on UNIX-like operating systems or `>` NUL on Windows. This way the process will not hang even if there would be a lot of output, but naturally output is not available after execution either.

Support for the special value `DEVNULL` is new in Robot Framework 3.2.

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

== Output encoding ==

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

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

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

== Alias ==

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

== Active process ==

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

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

== Result object ==

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

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

True examples:

False examples:

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

= Example =

```python
ROBOT_LIBRARY_SCOPE = 'GLOBAL'
ROBOT_LIBRARY_VERSION = '3.2b3.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.
start_process (command, *arguments, **configuration)
    Starts a new process on background.
    See Specifying command and arguments and Process configuration for more information about the arguments, and Run Process keyword for related examples.
    Makes the started process new active process. Returns an identifier that can be used as a handle to activate the started process if needed.
    Processes are started so that they create a new process group. This allows sending signals to and terminating also possible child processes. This is not supported on Jython.
is_process_running (handle=None)
    Checks is the process running or not.
    If handle is not given, uses the current active process.
```
Returns True if the process is still running and False otherwise.

**process_should_be_running** *(handle=None, error_message='Process is not running.‘)*
Verifies that the process is running.
If handle is not given, uses the current active process.
Fails if the process has stopped.

**process_should_be_stopped** *(handle=None, error_message='Process is running.‘)*
Verifies that the process is not running.
If handle is not given, uses the current active process.
Fails if the process is still running.

**wait_for_process** *(handle=None, timeout=None, on_timeout='continue‘)*
Waits for the process to complete or to reach the given timeout.
The process to wait for must have been started earlier with Start Process. If handle is not given, uses the current active process.

timeout defines the maximum time to wait for the process. It can be given in [http://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#time-format various time formats] supported by Robot Framework, for example, 42, 42 s, or 1 minute 30 seconds. The timeout is ignored if it is Python None (default), string NONE (case-insensitively), zero, or negative.

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.

Ignoring timeout if it is string NONE, zero, or negative is new in Robot Framework 3.2.

**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 when using Jython.
- Process is killed instead.
  - Stopping the whole process group is not supported when using Jython.
  - On Windows forceful kill only stops the main process, not possible child processes.
**terminate_all_processes** (*kill=False*)
Terminates all still running processes started by this library.
This keyword can be used in suite teardown or elsewhere to make sure that all processes are stopped.
By default tries to terminate processes gracefully, but can be configured to forcefully kill them immediately.
See **Terminate Process** that this keyword uses internally for more details.

**send_signal_to_process** (*signal*, *handle=None*, *group=False*)
Sends the given *signal* to the specified process.
If *handle* is not given, uses the current active process.
Signal can be specified either as an integer as a signal name. In the latter case it is possible to give the name both with or without SIG prefix, but names are case-sensitive. For example, all the examples below send signal INT (2):
This keyword is only supported on Unix-like machines, not on Windows. What signals are supported depends on the system. For a list of existing signals on your system, see the Unix man pages related to signal handling (typically *man signal* or *man 7 signal*).
By default sends the signal only to the parent process, not to possible child processes started by it. Notice that when running processes in shell, the shell is the parent process and it depends on the system does the shell propagate the signal to the actual started process.
To send the signal to the whole process group, *group* argument can be set to any true value (see *Boolean arguments*). This is not supported by Jython, however.

**get_process_id** (*handle=None*)
Returns the process ID (pid) of the process as an integer.
If *handle* is not given, uses the current active process.
Notice that the pid is not the same as the handle returned by **Start Process** that is used internally by this library.

**get_process_object** (*handle=None*)
Return the underlying *subprocess.Popen* object.
If *handle* is not given, uses the current active process.

**get_process_result** (*handle=None*, *rc=False*, *stdout=False*, *stderr=False*, *stdout_path=False*, *stderr_path=False*)
Returns the specified *result object* or some of its attributes.
The given *handle* specifies the process whose results should be returned. If no *handle* is given, results of the current active process are returned. In either case, the process must have been finishes before this keyword can be used. In practice this means that processes started with **Start Process** must be finished either with **Wait For Process** or **Terminate Process** before using this keyword.
If no other arguments than the optional *handle* are given, a whole *result object* is returned. If one or more of the other arguments are given any true value, only the specified attributes of the *result object* are returned. These attributes are always returned in the same order as arguments are specified in the keyword signature. See **Boolean arguments** section for more details about true and false values.
Although getting results of a previously executed process can be handy in general, the main use case for this keyword is returning results over the remote library interface. The remote interface does not support returning the whole result object, but individual attributes can be returned without problems.

**switch_process** (*handle*)
Makes the specified process the current active process.
The handle can be an identifier returned by **Start Process** or the alias given to it explicitly.

---

4.1. robot package 75
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

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

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

The default timeout is typically several minutes, but it depends on the operating system and its configuration. Notice that setting a timeout that is shorter than keyword execution time will interrupt the keyword.
Timeouts do not work with IronPython.

ROBOT_LIBRARY_SCOPE = 'TEST SUITE'

get_keyword_names(attempts=2)
get_keyword_arguments(name)
get_keyword_types(name)
get_keyword_tags(name)
get_keyword_documentation(name)
run_keyword(name, args, kwargs)

class robot.libraries.Remote.ArgumentCoerce
    Bases: object
    binary = <_sre.SRE_Pattern object>
    non_ascii = <_sre.SRE_Pattern object>
    coerce(argument)

class robot.libraries.Remote.RemoteResult(result)
    Bases: object

class robot.libraries.Remote.XmlRpcRemoteClient(uri, timeout=None)
    Bases: object
    get_keyword_names()
    get_keyword_arguments(name)
    get_keyword_types(name)
    get_keyword_tags(name)
    get_keyword_documentation(name)
    run_keyword(name, args, kwargs)

class robot.libraries.Remote.TimeoutHTTPTransport(use_datetime=0, timeout=None)
    Bases: xmlrpclib.Transport
    make_connection(host)
    accept_gzip_encoding = True
    close()
    encode_threshold = None
    get_host_info(host)
    getparser()
    parse_response(response)
    request(host, handler, request_body, verbose=0)
    send_content(connection, request_body)
    send_host(connection, host)
    send_request(connection, handler, request_body)
    send_user_agent(connection)
    single_request(host, handler, request_body, verbose=0)
    user_agent = 'xmlrpclib.py/1.0.1 (by www.pythonware.com)'

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

4.1. robot package
encode_threshold = None
get_host_info (host)
getparser ()
make_connection (host)
parse_response (response)
request (host, handler, request_body, verbose=0)
send_content (connection, request_body)
send_host (connection, host)
send_request (connection, handler, request_body)
send_user_agent (connection)
single_request (host, handler, request_body, verbose=0)
user_agent = 'xmlrpclib.py/1.0.1 (by www.pythonware.com)'

robot.libraries.Reserved module

class robot.libraries.Reserved.Reserved
    Bases: object
    ROBOT_LIBRARY_SCOPE = 'GLOBAL'
    get_keyword_names ()
    run_keyword (name, args)

robot.libraries.Screenshot module

class robot.libraries.Screenshot.Screenshot (screenshot_directory=None, screenshot_module=None)
    Bases: object
    Test library for taking screenshots on the machine where tests are run.
    Notice that successfully taking screenshots requires tests to be run with a physical or virtual display.
    = Using with Python =
    How screenshots are taken when using Python depends on the operating system. On OSX screenshots are taken using the built-in screencapture utility. On other operating systems you need to have one of the following tools or Python modules installed. You can specify the tool/module to use when importing the library. If no tool or module is specified, the first one found will be used.
    • wxPython :: http://wxpython.org :: Required also by RIDE so many Robot Framework users already have this module installed.
    • PyGTK :: http://pygtk.org :: This module is available by default on most Linux distributions.
    • Pillow :: http://python-pillow.github.io :: Only works on Windows. Also the original PIL package is supported.
    • Scrot :: http://en.wikipedia.org/wiki/Scrot :: Not used on Windows. Install with apt-get install scrot or similar.
Using `screencapture` on OSX and specifying explicit screenshot module are new in Robot Framework 2.9.2. The support for using `scrot` is new in Robot Framework 3.0.

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.

ScreenCapLibrary

[https://github.com/mihaiparvu/ScreenCapLibrary|ScreenCapLibrary] is an external Robot Framework library that can be used as an alternative, which additionally provides support for multiple formats, adjusting the quality, using GIFs and video capturing.

Configure where screenshots are saved.

If `screenshot_directory` is not given, screenshots are saved into same directory as the log file. The directory can also be set using `Set Screenshot Directory` keyword.

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

Specifying explicit screenshot module is new in Robot Framework 2.9.2.

ROBOT_LIBRARY_SCOPE = 'TEST SUITE'

ROBOT_LIBRARY_VERSION = '3.2b3.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 = '3.2b3.dev1'

class convert_to_lowercase(string)
   Converts string to lowercase.

class convert_to_uppercase(string)
   Converts string to uppercase.

class encode_string_to_bytes(string, encoding, errors='strict')
   Encodes the given Unicode string to bytes using the given encoding.
   errors argument controls what to do if encoding some characters fails. All values accepted by encode method in Python are valid, but in practice the following values are most useful:
   • strict: fail if characters cannot be encoded (default)
   • ignore: ignore characters that cannot be encoded
   • replace: replace characters that cannot be encoded with a replacement character

   Use Convert To Bytes in BuiltIn if you want to create bytes based on character or integer sequences. Use Decode Bytes To String in BuiltIn 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 \( (\text{bytes, encoding, errors='strict'}) \)

Decodes the given bytes to a Unicode string using the given encoding.

errors argument controls what to do if decoding some bytes fails. All values accepted by decode method in Python are valid, but in practice the following values are most useful:

- **strict**: fail if characters cannot be decoded (default)
- **ignore**: ignore characters that cannot be decoded
- **replace**: replace characters that cannot be decoded with a replacement character

Use Encode String To Bytes if you need to convert Unicode strings to byte strings, and Convert To String in BuiltIn if you need to convert arbitrary objects to Unicode strings.

format_string \( (\text{template, *positional, **named}) \)

Formats a template using the given positional and named arguments.

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

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

New in Robot Framework 3.1.

get_line_count \( (\text{string}) \)

Returns and logs the number of lines in the given string.

split_to_lines \( (\text{string, start=0, end=None}) \)

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 \( (\text{string, line_number}) \)

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 \( (\text{string, pattern, case_insensitive=False}) \)

Returns lines of the given string that contain the pattern.

The pattern is always considered to be a normal string, not a glob or regexp pattern. A line matches if the pattern is found anywhere on it.

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

Lines are returned as one string catenated back together with newlines. Possible trailing newline is never returned. The number of matching lines is automatically logged.
See Get Lines Matching Pattern and Get Lines Matching Regexp if you need more complex pattern matching.

get_lines_matching_pattern(string, pattern, case_insensitive=False)
Returns lines of the given string that match the pattern.

The pattern is a _glob pattern_ where:

A line matches only if it matches the pattern fully.

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

Lines are returned as one string 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.

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

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

split_string (string, separator=None, max_split=-1)
Splits the string using separator as a delimiter string.

If a separator is not given, any whitespace string is a separator. In that case also possible consecutive whitespace as well as leading and trailing whitespace is ignored.

Split words are returned as a list. If the optional max_split is given, at most max_split splits are done, and the returned list will have maximum max_split + 1 elements.

See Split String From Right if you want to start splitting from right, and Fetch From Left and Fetch From Right if you only want to get first/last part of the string.

split_string_from_right (string, separator=None, max_split=-1)
Splits the string using separator starting from right.

Same as Split String, but splitting is started from right. This has an effect only when max_split is given.

split_string_to_characters (string)
Splits the given string to characters.

fetch_from_left (string, marker)
Returns contents of the string before the first occurrence of marker.

If the marker is not found, whole string is returned.

See also Fetch From Right, Split String and Split String From Right.

fetch_from_right (string, marker)
Returns contents of the string after the last occurrence of marker.
If the marker is not found, whole string is returned.

See also Fetch From Left, Split String and Split String From Right.

**generate_random_string** *(length=8, chars='[LETTERS][NUMBERS]')*

Generates a string with a desired length from the given chars.

The population sequence chars contains the characters to use when generating the random string. It can contain any characters, and it is possible to use special markers explained in the table below:

**get_substring** *(string, start, end=None)*

Returns a substring from start index to end index.

The start index is inclusive and end is exclusive. Indexing starts from 0, and it is possible to use negative indices to refer to characters from the end.

**strip_string** *(string, mode='both', characters=None)*

Remove leading and/or trailing whitespaces from the given string.

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

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

New in Robot Framework 3.0.

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

Fails if the given item is not a string.

With Python 2, except with IronPython, this keyword passes regardless is the item a Unicode string or a byte string. Use Should Be Unicode String or Should Be Byte String if you want to restrict the string type. Notice that with Python 2, except with IronPython, 'string' creates a byte string and u'unicode' must be used to create a Unicode string.

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

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

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

Fails if the given item is a string.

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

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

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

Fails if the given item is not a Unicode string.

Use Should Be Byte String if you want to verify the item is a byte string, or Should Be String if both Unicode and byte strings are fine. See Should Be String for more details about Unicode strings and byte strings.

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

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

Fails if the given item is not a byte string.

Use Should Be Unicode String if you want to verify the item is a Unicode string, or Should Be String if both Unicode and byte strings are fine. See Should Be String for more details about Unicode strings and byte strings.
The default error message can be overridden with the optional `msg` argument.

**should_be_lowercase** *(string, msg=None)*

Fails if the given string is not in lowercase.

For example, 'string' and 'with specials!' would pass, and 'String', ', ' and ' ' would fail.

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

See also *Should Be Uppercase* and *Should Be Titlecase*.

**should_be_uppercase** *(string, msg=None)*

Fails if the given string is not in uppercase.

For example, 'STRING' and 'WITH SPECIALS!' would pass, and 'String', ', ' and ' ' would fail.

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

See also *Should Be Titlecase* and *Should Be Lowercase*.

**should_be_titlecase** *(string, msg=None)*

Fails if given string is not title.

string is a titlecased string if there is at least one character in it, uppercase characters only follow uncased characters and lowercase characters only cased ones.

For example, 'This Is Title' would pass, and 'Word In UPPER', 'Word In lower', ', ' and ' ' would fail.

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

See also *Should Be Uppercase* and *Should Be Lowercase*.

### robot.libraries.Telnet module

**class robot.libraries.Telnet** *(timeout='3 seconds', newline='CRLF', prompt=None, prompt_is_regexp=False, encoding='UTF-8', encoding_errors='ignore', default_log_level='INFO', window_size=None, environ_user=None, terminal_emulation=False, terminal_type=None, telnetlib_log_level='TRACE', connection_timeout=None)*

**Bases:** `object`

A test library providing communication over Telnet connections.

**Telnet** is Robot Framework’s standard library that makes it possible to connect to Telnet servers and execute commands on the opened connections.

== Table of contents ==

- Connections
- Writing and reading
- Configuration
- Terminal emulation
- Logging
- Time string format
- Boolean arguments
• Importing
• Shortcuts
• Keywords

= Connections =

The first step of using Telnet is opening a connection with Open Connection keyword. Typically the next step is logging in with Login keyword, and in the end the opened connection can be closed with Close Connection.

It is possible to open multiple connections and switch the active one using Switch Connection. Close All Connections can be used to close all the connections, which is especially useful in suite teardowns to guarantee that all connections are always closed.

= Writing and reading =

After opening a connection and possibly logging in, commands can be executed or text written to the connection for other reasons using Write and Write Bare keywords. The main difference between these two is that the former adds a [Configuration]configurable newline after the text automatically.

After writing something to the connection, the resulting output can be read using Read, Read Until, Read Until Regexp, and Read Until Prompt keywords. Which one to use depends on the context, but the latest one is often the most convenient.

As a convenience when running a command, it is possible to use Execute Command that simply uses Write and Read Until Prompt internally. Write Until Expected Output is useful if you need to wait until writing something produces a desired output.

Written and read text is automatically encoded/decoded using a [Configuration]configured encoding.

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

= Configuration =

Many aspects related the connections can be easily configured either globally or per connection basis. Global configuration is done when [Importing]library is imported, and these values can be overridden per connection by Open Connection or with setting specific keywords Set Timeout, Set Newline, Set Prompt, Set Encoding, Set Default Log Level and Set Telnetlib Log Level.

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

== Timeout ==

Timeout defines how long is the maximum time to wait when reading output. It is used internally by Read Until, Read Until Regexp, Read Until Prompt, and Login keywords. The default value is 3 seconds.

== Connection Timeout ==

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

New in Robot Framework 2.9.2.

== Newline ==

Newline defines which line separator Write keyword should use. The default value is CRLF that is typically used by Telnet connections.

Newline can be given either in escaped format using \n and \r or with special LF and CR syntax.
== Prompt ==

Often the easiest way to read the output of a command is reading all the output until the next prompt with *Read Until Prompt*. It also makes it easier, and faster, to verify did *Login succeed*.

Prompt can be specified either as a normal string or a regular expression. The latter is especially useful if the prompt changes as a result of the executed commands. Prompt can be set to be a regular expression by giving `prompt_is_regexp` argument a true value (see *Boolean arguments*).

== Encoding ==

To ease handling text containing non-ASCII characters, all written text is encoded and read text decoded by default. The default encoding is UTF-8 that works also with ASCII. Encoding can be disabled by using a special encoding value `NONE`. This is mainly useful if you need to get the bytes received from the connection as-is.

Notice that when writing to the connection, only Unicode strings are encoded using the defined encoding. Byte strings are expected to be already encoded correctly. Notice also that normal text in test data is passed to the library as Unicode and you need to use variables to use bytes.

It is also possible to configure the error handler to use if encoding or decoding characters fails. Accepted values are the same that encode/decode functions in Python strings accept. In practice the following values are the most useful:

- **ignore**: ignore characters that cannot be encoded (default)
- **strict**: fail if characters cannot be encoded
- **replace**: replace characters that cannot be encoded with a replacement character

== Default log level ==

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

== Terminal type ==

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

== Window size ==

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

== USER environment variable ==

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

Terminal emulation =

Telnet library supports terminal emulation with [Pyte](http://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[3Dbb` 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`. 

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

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

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

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

### Logging

All keywords that read something log the output. These keywords take the log level to use as an optional argument, and if no log level is specified they use the [#Configuration|configured] default value.

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

The [http://docs.python.org/library/telnetlib.html|telnetlib module] used by this library has a custom logging system for logging content it sends and receives. By default these messages are written using `TRACE` level, but the level is configurable with the `telnetlib_log_level` option either in the library initialization, to the `Open Connection` or by using the `Set Telnetlib Log Level` keyword to the active connection. Special level `NONE` 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|Robot Framework User Guide].

### Boolean arguments

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

True examples:

False examples:

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

Telnet library can be imported with optional configuration parameters.

Configuration parameters are used as default values when new connections are opened with `Open Connection` keyword. They can also be overridden after opening the connection using the `Set ...` keywords. See these keywords as well as `Configuration`, `Terminal emulation` and `Logging` sections above for more information about these parameters and their possible values.

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

```
ROBOT_LIBRARY_SCOPE = 'TEST_SUITE'
ROBOT_LIBRARY_VERSION = '3.2b3.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 [#Importing|imported]. Setting them here overrides those values for the opened connection. See Configuration, Terminal emulation and Logging sections for more information about these parameters and their possible values.

Possible already opened connections are cached and it is possible to switch back to them using Switch Connection keyword. It is possible to switch either using explicitly given alias or using index returned by this keyword. Indexing starts from 1 and is reset back to it by Close All Connections keyword.

switch_connection (index_or_alias)

Switches between active connections using an index or an alias.

Aliases can be given to Open Connection keyword which also always returns the connection index.

This keyword returns the index of previous active connection.

The example above expects that there were no other open connections when opening the first one, because it used index 1 when switching to the connection later. If you are not sure about that, you can store the index into a variable as shown below.

close_all_connections()

Closes all open connections and empties the connection cache.

If multiple connections are opened, this keyword should be used in a test or suite teardown to make sure that all connections are closed. It is not an error if some of the connections have already been closed by Close Connection.

After this keyword, new indexes returned by Open Connection keyword are reset to 1.

class robot.libraries.Telnet.TelnetConnection (host=None, port=23, timeout=3.0, newline='CRLF', prompt=None, prompt_is_regexp=False, encoding='UTF-8', encoding_errors='ignore', default_log_level='INFO', window_size=None, environ_user=None, terminal_emulation=False, terminal_type=None, telnetlib_log_level='TRACE', connection_timeout=None)

Bases: telnetlib.Telnet

NEW_ENVIRON_IS = '\x00'

NEW_ENVIRON_VAR = '\x00'

NEW_ENVIRON_VALUE = '\x01'

INTERNAL_UPDATE_FREQUENCY = 0.03

set_timeout (timeout)

Sets the timeout used for waiting output in the current connection.
Read operations that expect some output to appear (Read Until, Read Until Regexp, Read Until Prompt, Login) use this timeout and fail if the expected output does not appear before this timeout expires.

The timeout must be given in time string format. The old timeout is returned and can be used to restore the timeout later.

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

**set_newline** (newline)
Sets the newline used by Write keyword in the current connection.

The old newline is returned and can be used to restore the newline later. See Set Timeout for a similar example.

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

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

**set_prompt** (prompt, prompt_is_regexp=False)
Sets the prompt used by Read Until Prompt and Login in the current connection.

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

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

See the documentation of [http://docs.python.org/library/re.html|Python re module] for more information about the supported regular expression syntax. Notice that possible backslashes need to be escaped in Robot Framework test data.

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

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

The given encoding specifies the encoding to use when written/read text is encoded/decoded, and errors specifies the error handler to use if encoding/decoding fails. Either of these can be omitted and in that case the old value is not affected. Use string NONE to disable encoding altogether.

See Configuration section for more information about encoding and error handlers, as well as global and connection specific configuration in general.

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

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

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

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

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

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

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

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

Remaining output in the connection is read, logged, and returned. It is not an error to close an already closed connection.
Use **Close All Connections** if you want to make sure all opened connections are closed.

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

`login(username, password, login_prompt='login: ', password_prompt='Password: ', login_timeout='1 second', login_incorrect='Login incorrect')`

Logs in to the Telnet server with the given user information.

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

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

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

2) If the prompt is not set, this keywords sleeps until `login_timeout` and then reads all the output available on the connection. If the output contains `login_incorrect` text, login is considered failed and also this keyword fails.

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

`write(text, loglevel=None)`

Writes the given text plus a newline into the connection.

The newline character sequence to use can be [#Configuration|configured] both globally and per connection basis. The default value is CRLF.

This keyword consumes the written text, until the added newline, from the output and logs and returns it. The given text itself must not contain newlines. Use `Write Bare` instead if either of these features causes a problem.

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

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

`write_bare(text)`

Writes the given text, and nothing else, into the connection.

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

`write_until_expected_output(text, expected, timeout, retry_interval, loglevel=None)`

Writes the given text repeatedly, until expected appears in the output.

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

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

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

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

`write_control_character(character)`

Writes the given control character into the connection.
The control character is prepended with an IAC (interpret as command) character.

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

`read(loglevel=None)`
Reads everything that is currently available in the output.
Read output is both returned and logged. See Logging section for more information about log levels.

`read_until(expected, loglevel=None)`
Reads output until expected text is encountered.
Text up to and including the match is returned and logged. If no match is found, this keyword fails. How much to wait for the output depends on the [#Configuration|configured timeout].

See Logging section for more information about log levels. Use Read Until Regexp if more complex matching is needed.

`read_until_regexp(*expected)`
Reads output until any of the expected regular expressions match.
This keyword accepts any number of regular expressions patterns or compiled Python regular expression objects as arguments. Text up to and including the first match to any of the regular expressions is returned and logged. If no match is found, this keyword fails. How much to wait for the output depends on the [#Configuration|configured timeout].

If the last given argument is a [#Logging|valid log level], it is used as loglevel similarly as with Read Until keyword.

See the documentation of [http://docs.python.org/library/re.html|Python re module] for more information about the supported regular expression syntax. Notice that possible backslashes need to be escaped in Robot Framework test data.

`read_until_prompt(loglevel=None, strip_prompt=False)`
Reads output until the prompt is encountered.
This keyword requires the prompt to be [#Configuration|configured] either in importing or with Open Connection or Set Prompt keyword.
By default, text up to and including the prompt is returned and logged. If no prompt is found, this keyword fails. How much to wait for the output depends on the [#Configuration|configured timeout].
If you want to exclude the prompt from the returned output, set strip_prompt to a true value (see Boolean arguments). If your prompt is a regular expression, make sure that the expression spans the whole prompt, because only the part of the output that matches the regular expression is stripped away.

See Logging section for more information about log levels.

`execute_command(command, loglevel=None, strip_prompt=False)`
Executes the given command and reads, logs, and returns everything until the prompt.
This keyword requires the prompt to be [#Configuration|configured] either in importing or with Open Connection or Set Prompt keyword.
This is a convenience keyword that uses Write and Read Until Prompt internally. Following two examples are thus functionally identical:

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

`msg(msg, *args)`
close()
Close the connection.

expect (list, timeout=None)
Read until one from a list of a regular expressions matches.

The first argument is a list of regular expressions, either compiled (re.RegexObject instances) or uncompiled (strings). The optional second argument is a timeout, in seconds; default is no timeout.

Return a tuple of three items: the index in the list of the first regular expression that matches; the match object returned; and the text read up till and including the match.

If EOF is read and no text was read, raise EOFError. Otherwise, when nothing matches, return (-1, None, text) where text is the text received so far (may be the empty string if a timeout happened).

If a regular expression ends with a greedy match (e.g. `.`) or if more than one expression can match the same input, the results are undeterministic, and may depend on the I/O timing.

fileno()
Return the fileno() of the socket object used internally.

close()
Fill raw queue from exactly one recv() system call.

Block if no data is immediately available. Set self.eof when connection is closed.

get_socket()
Return the socket object used internally.

interact()
Interaction function, emulates a very dumb telnet client.

listener()
Helper for mt_interact() – this executes in the other thread.

mt_interact()
Multithreaded version of interact().

open (host, port=0, timeout=<object object>)
Connect to a host.

The optional second argument is the port number, which defaults to the standard telnet port (23).

Don’t try to reopen an already connected instance.

process_rawq()
Transfer from raw queue to cooked queue.

Set self.eof when connection is closed. Don’t block unless in the midst of an IAC sequence.

rawq_getchar()
Get next char from raw queue.

Block if no data is immediately available. Raise EOFError when connection is closed.

read_all()
Read all data until EOF; block until connection closed.

read_eager()
Read readily available data.

Raise EOFError if connection closed and no cooked data available. Return ‘’ if no cooked data available otherwise. Don’t block unless in the midst of an IAC sequence.
**Robot Framework Documentation, Release 3.2b3.dev1**

`read_lazy()`  
Process and return data that’s already in the queues (lazy).  
Raise EOFError if connection closed and no data available. Return ‘’ if no cooked data available otherwise.  
Don’t block unless in the midst of an IAC sequence.

`read_sb_data()`  
Return any data available in the SB ... SE queue.  
Return ‘’ if no SB ... SE available. Should only be called after seeing a SB or SE command. When a new SB command is found, old unread SB data will be discarded.  
Don’t block.

`read_some()`  
Read at least one byte of cooked data unless EOF is hit.  
Return ‘’ if EOF is hit. Block if no data is immediately available.

`read_very_eager()`  
Read everything that’s possible without blocking in I/O (eager).  
Raise EOFError if connection closed and no cooked data available. Return ‘’ if no cooked data available otherwise.  
Don’t block unless in the midst of an IAC sequence.

`read_very_lazy()`  
Return any data available in the cooked queue (very lazy).  
Raise EOFError if connection closed and no data available. Return ‘’ if no cooked data available otherwise.  
Don’t block.

`set_debuglevel(debuglevel) `  
Set the debug level.  
The higher it is, the more debug output you get (on sys.stdout).

`set_option_negotiation_callback(callback) `  
Provide a callback function called after each receipt of a telnet option.

`sock_avail()`  
Test whether data is available on the socket.

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

`current_output`  
feed(text)

`read()`  
`read_until(expected)`

`read_until_regexp(regex_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 [ElementTree XML API](http://docs.python.org/library/xml.etree.elementtree.html).

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
- Element attributes
- Handling XML namespaces
- Boolean arguments
- Pattern matching
- Shortcuts
- Keywords

= Parsing XML =

XML can be parsed into an element structure using Parse XML keyword. The XML to be parsed can be specified using a path to an XML file or as a string or bytes that contain XML directly. The keyword returns the root element of the structure, which then contains other elements as its children and their children. Possible comments and processing instructions in the source XML are removed.

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

The element structure returned by Parse XML, as well as elements returned by keywords such as Get Element, can be used as the source argument with other keywords. In addition to an already parsed XML structure, other keywords also accept paths to XML files and strings containing XML similarly as Parse XML. Notice that keywords that modify XML do not write those changes back to disk even if the source would be given as a path to a file. Changes must always saved explicitly using Save XML keyword.

When the source is given as a path to a file, the forward slash character (/) can be used as the path separator regardless the operating system. On Windows also the backslash works, but it the test data it needs to be escaped by doubling it (\). Using the built-in variable `$()` naturally works too.

Note: Support for XML as bytes is new in Robot Framework 3.2.
= Using lxml =

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

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

= Example =

The following simple example demonstrates parsing XML and verifying its contents both using keywords in this library and in _BuiltIn_ and _Collections_ libraries. How to use xpath expressions to find elements and what attributes the returned elements contain are discussed, with more examples, in Finding elements with xpath and Element attributes sections.

In this example, as well as in many other examples in this documentation, ${XML} refers to the following example XML document. In practice ${XML} could either be a path to an XML file or it could contain the XML itself.

Notice that in the example three last lines are equivalent. Which one to use in practice depends on which other elements you need to get or verify. If you only need to do one verification, using the last line alone would suffice. If more verifications are needed, parsing the XML with Parse XML only once would be more efficient.

= Finding elements with xpath =

ElementTree, and thus also this library, supports finding elements using xpath expressions. ElementTree does not, however, support the full xpath standard. The supported xpath syntax is explained below and [https://docs.python.org/library/xml.etree.elementtree.html#xpath-support|ElementTree documentation] provides more details. In the examples ${XML} refers to the same XML structure as in the earlier example.

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

== Tag names ==

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

== Paths ==

Paths are created by combining tag names with a forward slash (/). For example, parent/child matches all child elements under parent element. Notice that if there are multiple parent elements that all have child elements, parent/child xpath will match all these child elements.

== Wildcards ==

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

== Current element ==

The current element is denoted with a dot (.). Normally the current element is implicit and does not need to be included in the xpath.

== Parent element ==

The parent element of another element is denoted with two dots (..). Notice that it is not possible to refer to the parent of the current element.

== Search all sub elements ==

Two forward slashes (//) mean that all sub elements, not only the direct children, are searched. If the search is started from the current element, an explicit dot is required.

== Predicates ==
Predicates allow selecting elements using also other criteria than tag names, for example, attributes or position. They are specified after the normal tag name or path using syntax `path[predicate]`. The path can have wildcards and other special syntax explained earlier. What predicates the standard `ElementTree` supports is explained in the table below.

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

= Element attributes =

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

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

The examples use the same `$XML` structure as the earlier examples.

== tag ==

The tag of the element.

== text ==

The text that the element contains or Python `None` if the element has no text. Notice that the text _does not_ contain texts of possible child elements nor text after or between children. Notice also that in XML whitespace is significant, so the text contains also possible indentation and newlines. To get also text of the possible children, optionally whitespace normalized, use `Get Element Text` keyword.

== tail ==

The text after the element before the next opening or closing tag. Python `None` if the element has no tail. Similarly as with `text`, `also tail` contains possible indentation and newlines.

== attrib ==

A Python dictionary containing attributes of the element.

= Handling XML namespaces =

`ElementTree` and `lxml` handle possible namespaces in XML documents by adding the namespace URI to tag names in so called Clark Notation. That is inconvenient especially with xpaths, and by default this library strips those namespaces away and moves them to `xmlns` attribute instead. That can be avoided by passing `keep_clark_notation` argument to `Parse XML` keyword. Alternatively `Parse XML` supports stripping namespace information altogether by using `strip_namespaces` argument. The pros and cons of different approaches are discussed in more detail below.

== How `ElementTree` handles namespaces ==

If an XML document has namespaces, `ElementTree` adds namespace information to tag names in [http://www.jclark.com/xml/xmlns.html|Clark Notation] (e.g. `{http://ns.uri}tag`) and removes original `xmlns` attributes. This is done both with default namespaces and with namespaces with a prefix. How it works in practice is illustrated by the following example, where `$NS` variable contains this XML document:

As you can see, including the namespace URI in tag names makes xpaths really long and complex.

If you save the XML, `ElementTree` moves namespace information back to `xmlns` attributes. Unfortunately it does not restore the original prefixes:

The resulting output is semantically same as the original, but mangling prefixes like this may still not be desirable. Notice also that the actual output depends slightly on `ElementTree` version.
== Default namespace handling ==

Because the way ElementTree handles namespaces makes xpaths so complicated, this library, by default, strips namespaces from tag names and moves that information back to xmlns attributes. How this works in practice is shown by the example below, where ${NS} variable contains the same XML document as in the previous example.

Now that tags do not contain namespace information, xpaths are simple again.

A minor limitation of this approach is that namespace prefixes are lost. As a result the saved output is not exactly same as the original one in this case either:

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

== Namespaces when using lxml ==

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

== Stripping namespaces altogether ==

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

== Attribute namespaces ==

Attributes in XML documents are, by default, in the same namespaces as the element they belong to. It is possible to use different namespaces by using prefixes, but this is pretty rare.

If an attribute has a namespace prefix, ElementTree will replace it with Clark Notation the same way it handles elements. Because stripping namespaces from attributes could cause attribute conflicts, this library does not handle attribute namespaces at all. Thus the following example works the same way regardless how namespaces are handled.

== Boolean arguments ==

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

True examples:

False examples:

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

== Pattern matching ==

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

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

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

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

Using lxml requires that the lxml module is installed on the system. If lxml mode is enabled but the module is not installed, this library will emit a warning and revert back to using the standard ElementTree.

ROBOT_LIBRARY_SCOPE = 'GLOBAL'

ROBOT_LIBRARY_VERSION = '3.2b3.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 [http://docs.python.org/library/xml.etree.elementtree.html#element-objects|element structure] and the root element is returned. Possible comments and processing instructions in the source XML are removed.

As discussed in Handling XML namespaces section, this keyword, by default, removes namespace information ElementTree has added to tag names and moves it into xmlns attributes. This typically eases handling XML documents with namespaces considerably. If you do not want that to happen, or want to avoid the small overhead of going through the element structure when your XML does not have namespaces, you can disable this feature by giving keep_clark_notation argument a true value (see Boolean arguments).

If you want to strip namespace information altogether so that it is not included even if XML is saved, you can give a true value to strip_namespaces argument. This functionality is new in Robot Framework 3.0.2.

Use Get Element keyword if you want to get a certain element and not the whole structure. See Parsing XML section for more details and examples.

get_element (source, xpath='.')

Returns an element in the source matching the xpath.

The source can be a path to an XML file, a string containing XML, or an already parsed XML element. The xpath specifies which element to find. See the introduction for more details about both the possible sources and the supported xpath syntax.

The keyword fails if more, or less, than one element matches the xpath. Use Get Elements if you want all matching elements to be returned.

Parse XML is recommended for parsing XML when the whole structure is needed. It must be used if there is a need to configure how XML namespaces are handled.

Many other keywords use this keyword internally, and keywords modifying XML are typically documented to both to modify the given source and to return it. Modifying the source does not apply if the source is given as a string. The XML structure parsed based on the string and then modified is nevertheless returned.

get_elements (source, xpath)

Returns a list of elements in the source matching the xpath.

The source can be a path to an XML file, a string containing XML, or an already parsed XML element. The xpath specifies which element to find. See the introduction for more details.

Elements matching the xpath are returned as a list. If no elements match, an empty list is returned. Use Get Element if you want to get exactly one match.

get_child_elements (source, xpath='.')

Returns the child elements of the specified element as a list.
The element whose children to return is specified using `source` and `xpath`. They have exactly the same semantics as with `Get Element` keyword.

All the direct child elements of the specified element are returned. If the element has no children, an empty list is returned.

`get_element_count (source, xpath='.')`
Returns and logs how many elements the given `xpath` matches.
Arguments `source` and `xpath` have exactly the same semantics as with `Get Elements` keyword that this keyword uses internally.

See also `Element Should Exist` and `Element Should Not Exist`.

`element_should_exist (source, xpath='.', message=None)`
Verifies that one or more element match the given `xpath`.
Arguments `source` and `xpath` have exactly the same semantics as with `Get Elements` keyword. Keyword passes if the `xpath` matches one or more elements in the `source`. The default error message can be overridden with the `message` argument.

See also `Element Should Not Exist` as well as `Get Element Count` that this keyword uses internally.

`element_should_not_exist (source, xpath='.', message=None)`
Verifies that no element match the given `xpath`.
Arguments `source` and `xpath` have exactly the same semantics as with `Get Elements` keyword. Keyword fails if the `xpath` matches any element in the `source`. The default error message can be overridden with the `message` argument.

See also `Element Should Exist` as well as `Get Element Count` that this keyword uses internally.

`get_element_text (source, xpath='.', normalize_whitespace=False)`
Returns all text of the element, possibly whitespace normalized.
The element whose text to return is specified using `source` and `xpath`. They have exactly the same semantics as with `Get Element` keyword.

This keyword returns all the text of the specified element, including all the text its children and grandchildren contain. If the element has no text, an empty string is returned. The returned text is thus not always the same as the `text` attribute of the element.

By default all whitespace, including newlines and indentation, inside the element is returned as-is. If `normalize_whitespace` is given a true value (see Boolean arguments), then leading and trailing whitespace is stripped, newlines and tabs converted to spaces, and multiple spaces collapsed into one. This is especially useful when dealing with HTML data.

See also `Get Elements Texts`, `Element Text Should Be` and `Element Text Should Match`.

`get_elements_texts (source, xpath, normalize_whitespace=False)`
Returns text of all elements matching `xpath` as a list.
The elements whose text to return is specified using `source` and `xpath`. They have exactly the same semantics as with `Get Elements` keyword.

The text of the matched elements is returned using the same logic as with `Get Element Text`. This includes optional whitespace normalization using the `normalize_whitespace` option.

`element_text_should_be (source, expected, xpath='.', normalize_whitespace=False, message=None)`
Verifies that the text of the specified element is `expected`.
The element whose text is verified is specified using `source` and `xpath`. They have exactly the same semantics as with `Get Element` keyword.
The text to verify is got from the specified element using the same logic as with Get Element Text. This includes optional whitespace normalization using the normalize_whitespace option.

The keyword passes if the text of the element is equal to the expected value, and otherwise it fails. The default error message can be overridden with the message argument. Use Element Text Should Match to verify the text against a pattern instead of an exact value.

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

```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.

```python
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.

```python
element_attribute_should_match(source, name, pattern, xpath='.', message=None)
```

Verifies that the specified attribute matches expected.

This keyword works exactly like Element Attribute Should Be except that the expected value can be given as a pattern that the attribute of the element must match.

Pattern matching is similar as matching files in a shell with *, ?, and [chars] acting as wildcards. See the Pattern matching section for more information.
element_should_not_have_attribute(source, name=xpath='.', message=None)

Verifies that the specified element does not have attribute name.

The element whose attribute is verified is specified using source and xpath. They have exactly the same semantics as with Get Element keyword.

The keyword fails if the specified element has attribute name. The default error message can be overridden with the message argument.

See also Get Element Attribute, Get Element Attributes, Element Text Should Be and Element Text Should Match.

elements_should_be_equal(source, expected, exclude_children=False, normalize_whitespace=False)

Verifies that the given source element is equal to expected.

Both source and expected can be given as a path to an XML file, as a string containing XML, or as an already parsed XML element structure. See introduction for more information about parsing XML in general.

The keyword passes if the source element and expected element are equal. This includes testing the tag names, texts, and attributes of the elements. By default also child elements are verified the same way, but this can be disabled by setting exclude_children to a true value (see Boolean arguments).

All texts inside the given elements are verified, but possible text outside them is not. By default texts must match exactly, but setting normalize_whitespace to a true value makes text verification independent on newlines, tabs, and the amount of spaces. For more details about handling text see Get Element Text keyword and discussion about elements' text and tail attributes in the introduction.

The last example may look a bit strange because the <p> element only has text Text with. The reason is that rest of the text inside <p> actually belongs to the child elements. This includes the . at the end that is the tail text of the <i> element.

See also Elements Should Match.

elements_should_match(source, expected, exclude_children=False, normalize_whitespace=False)

Verifies that the given source element matches expected.

This keyword works exactly like Elements Should Be Equal except that texts and attribute values in the expected value can be given as patterns.

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

See Elements Should Be Equal for more examples.

set_element_tag(source, tag, xpath='.'

Sets the tag of the specified element.

The element whose tag to set is specified using source and xpath. They have exactly the same semantics as with Get Element keyword. The resulting XML structure is returned, and if the source is an already parsed XML structure, it is also modified in place.

Can only set the tag of a single element. Use Set Elements Tag to set the tag of multiple elements in one call.

set_elements_tag(source, tag, xpath='.'

Sets the tag of the specified elements.

Like Set Element Tag but sets the tag of all elements matching the given xpath.

set_element_text(source, text=None, tail=None, xpath='.'

Sets text and/or tail text of the specified element.
The element whose text to set is specified using `source` and `xpath`. They have exactly the same semantics as with `Get Element` keyword. The resulting XML structure is returned, and if the `source` is an already parsed XML structure, it is also modified in place.

Element’s text and tail text are changed only if new `text` and/or `tail` values are given. See `Element attributes` section for more information about `text` and `tail` in general.

Can only set the text/tail of a single element. Use `Set Elements Text` to set the text/tail of multiple elements in one call.

```python
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`.

```python
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.

```python
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`.

```python
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.

```python
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`.

```python
remove_element_attributes(source, xpath='.')
```

Removes all attributes from the specified element.

The element whose attributes to remove is specified using `source` and `xpath`. They have exactly the same semantics as with `Get Element` keyword. The resulting XML structure is returned, and if the `source` is an already parsed XML structure, it is also modified in place.

Use `Remove Element Attribute` to remove a single attribute and `Set Element Attribute` to set them.

Can only remove attributes from a single element. Use `Remove Elements Attributes` to remove all attributes of multiple elements in one call.
**remove_elements_attributes** `(source, xpath='')`

Removes all attributes from the specified elements.

Like **Remove Element Attributes** but removes all attributes of all elements matching the given `xpath`.

**add_element** `(source, element, index=None, xpath='')`

Adds a child element to the specified element.

The element to whom to add the new element is specified using `source` and `xpath`. They have exactly the same semantics as with **Get Element** keyword. The resulting XML structure is returned, and if the `source` is an already parsed XML structure, it is also modified in place.

The element to add can be specified as a path to an XML file or as a string containing XML, or it can be an already parsed XML element. The element is copied before adding so modifying either the original or the added element has no effect on the other. The element is added as the last child by default, but a custom index can be used to alter the position. Indices start from zero (0 = first position, 1 = second position, etc.), and negative numbers refer to positions at the end (-1 = second last position, -2 = third last, etc.).

Use **Remove Element** or **Remove Elements** to remove elements.

**remove_element** `(source, xpath=' ', remove_tail=False)`

Removes the element matching `xpath` from the `source` structure.

The element to remove from the `source` is specified with `xpath` using the same semantics as with **Get Element** keyword. The resulting XML structure is returned, and if the `source` is an already parsed XML structure, it is also modified in place.

The keyword fails if `xpath` does not match exactly one element. Use **Remove Elements** to remove all matched elements.

Element’s tail text is not removed by default, but that can be changed by giving `remove_tail` a true value (see **Boolean arguments**). See **Element attributes** section for more information about `tail` in general.

**remove_elements** `(source, xpath=' ', remove_tail=False)`

Removes all elements matching `xpath` from the `source` structure.

The elements to remove from the `source` are specified with `xpath` using the same semantics as with **Get Elements** keyword. The resulting XML structure is returned, and if the `source` is an already parsed XML structure, it is also modified in place.

It is not a failure if `xpath` matches no elements. Use **Remove Element** to remove exactly one element.

Element’s tail text is not removed by default, but that can be changed by using `remove_tail` argument similarly as with **Remove Element**.

**clear_element** `(source, xpath=' ', clear_tail=False)`

Clears the contents of the specified element.

The element to clear is specified using `source` and `xpath`. They have exactly the same semantics as with **Get Element** keyword. The resulting XML structure is returned, and if the `source` is an already parsed XML structure, it is also modified in place.

Clearing the element means removing its text, attributes, and children. Element’s tail text is not removed by default, but that can be changed by giving `clear_tail` a true value (see **Boolean arguments**). See **Element attributes** section for more information about tail in general.

Use **Remove Element** to remove the whole element.

**copy_element** `(source, xpath=' ')`

Returns a copy of the specified element.
The element to copy is specified using source and xpath. They have exactly the same semantics as with Get Element keyword.

If the copy or the original element is modified afterwards, the changes have no effect on the other.

element_to_string (source, xpath='.', encoding=None)

Returns the string representation of the specified element.

The element to convert to a string is specified using source and xpath. They have exactly the same semantics as with Get Element keyword.

By default the string is returned as Unicode. If encoding argument is given any value, the string is returned as bytes in the specified encoding. The resulting string never contains the XML declaration.

See also Log Element and Save XML.

log_element (source, level='INFO', xpath='.')

Logs the string representation of the specified element.

The element specified with source and xpath is first converted into a string using Element To String keyword internally. The resulting string is then logged using the given level.

The logged string is also returned.

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

Saves the given element to the specified file.

The element to save is specified with source using the same semantics as with Get Element keyword.

The file where the element is saved is denoted with path and the encoding to use with encoding. The resulting file always contains the XML declaration.

The resulting XML file may not be exactly the same as the original: - Comments and processing instructions are always stripped. - Possible doctype and namespace prefixes are only preserved when using lxml.

- Other small differences are possible depending on the ElementTree or lxml version.

Use Element To String if you just need a string representation of the element.

evaluate_xpath (source, expression, context='.')

Evaluates the given xpath expression and returns results.

The element in which context the expression is executed is specified using source and context arguments. They have exactly the same semantics as source and xpath arguments have with Get Element keyword.

The xpath expression to evaluate is given as expression argument. The result of the evaluation is returned as-is.

This keyword works only if lxml mode is taken into use when importing the library.

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

Bases: object

strip (elem, preserve=True, current_ns=None, top=True)

unstrip (elem, current_ns=None, copied=False)

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

Bases: object

find_all (elem, xpath)
class robot.libraries.XML.ElementComparator (comparator, normalizer=None, exclude_children=False)

Bases: object

compare (actual, expected, location=None)

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

Bases: object

child (tag)

robot.libraries.dialogs_ipy module

robot.libraries.dialogs_jy module

robot.libraries.dialogs_py module

class robot.libraries.dialogs_py.MessageDialog (message, value=None, **extra)

Bases: robot.libraries.dialogs_py._TkDialog

after (ms, func=None, *args)

Call function once after given time.

MS specifies the time in milliseconds. FUNC gives the function which shall be called. Additional parameters are given as parameters to the function call. Return identifier to cancel scheduling with after_cancel.

after_cancel (id)

Cancel scheduling of function identified with ID.

Identifier returned by after or after_idle must be given as first parameter.

after_idle (func, *args)

Call FUNC once if the Tcl main loop has no event to process.

Return an identifier to cancel the scheduling with after_cancel.

aspect (minNumer=None, minDenom=None, maxNumer=None, maxDenom=None)

Instruct the window manager to set the aspect ratio (width/height) of this widget to be between MINNUMER/MINDENOM and MAXNUMER/MAXDENOM. Return a tuple of the actual values if no argument is given.

attributes (*args)

This subcommand returns or sets platform specific attributes

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

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

On Macintosh, XXXXX

On Unix, there are currently no special attribute values.

bbox (column=None, row=None, col2=None, row2=None)

Return a tuple of integer coordinates for the bounding box of this widget controlled by the geometry manager grid.

If COLUMN, ROW is given the bounding box applies from the cell with row and column 0 to the specified cell. If COL2 and ROW2 are given the bounding box starts at that cell.
The returned integers specify the offset of the upper left corner in the master widget and the width and height.

**bell**\(\text{(displayof=0)}\)

Ring a display’s bell.

**bind**\(\text{(sequence=None, func=None, add=None)}\)

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**\(\text{(sequence=None, func=None, add=None)}\)

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**\(\text{(className, sequence=None, func=None, add=None)}\)

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**\(\text{(tagList=None)}\)

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**\(\text{(key)}\)

Return the resource value for a KEY given as string.

**client**\(\text{(name=None)}\)

Store NAME in WM_CLIENT_MACHINE property of this widget. Return current value.

**clipboard_append**\(\text{(string, **kw)}\)

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**\(\text{(**kw)}\)

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.

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

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

getdouble
alias of __builtin__.float

getint
alias of __builtin__.int

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

grab_current()
Return widget which has currently the grab in this application or None.
grab_release()
Release grab for this widget if currently set.

grab_set (timeout=30)

grab_set_global()
Set global grab for this widget.

A global grab directs all events to this and descendant widgets on the display. Use with caution - other applications do not get events anymore.

grab_status()
Return None, “local” or “global” if this widget has no, a local or a global grab.

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

grid_bbox (column=None, row=None, col2=None, row2=None)
Return a tuple of integer coordinates for the bounding box of this widget controlled by the geometry manager grid.

If COLUMN, ROW is given the bounding box applies from the cell with row and column 0 to the specified cell. If COL2 and ROW2 are given the bounding box starts at that cell.

The returned integers specify the offset of the upper left corner in the master widget and the width and height.

grid_columnconfigure (index, cnf={}, **kw)
Configure column INDEX of a grid.

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

grid_location (x, y)
Return a tuple of column and row which identify the cell at which the pixel at position X and Y inside the master widget is located.

grid_propagate (flag=[‘_noarg_’])
Set or get the status for propagation of geometry information.

A boolean argument specifies whether the geometry information of the slaves will determine the size of this widget. If no argument is given, the current setting will be returned.

grid_rowconfigure (index, cnf={}, **kw)
Configure row INDEX of a grid.

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

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

grid_slaves (row=None, column=None)
Return a list of all slaves of this widget in its packing order.

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

iconbitmap (bitmap=None, default=None)
Set bitmap for the iconified widget to BITMAP. Return the bitmap if None is given.
Under Windows, the DEFAULT parameter can be used to set the icon for the widget and any 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.

override_redirect (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**(wargs)

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.
wininfo_reqheight ()
Return requested height of this widget.

wininfo_reqwidth ()
Return requested width of this widget.

wininfo_rgb (color)
Return tuple of decimal values for red, green, blue for COLOR in this widget.

wininfo_rootx ()
Return x coordinate of upper left corner of this widget on the root window.

wininfo_rooty ()
Return y coordinate of upper left corner of this widget on the root window.

wininfo_screen ()
Return the screen name of this widget.

wininfo_screencells ()
Return the number of the cells in the colormap of the screen of this widget.

wininfo_screendepth ()
Return the number of bits per pixel of the root window of the screen of this widget.

wininfo_screenheight ()
Return the number of pixels of the height of the screen of this widget in pixel.

wininfo_screennmheight ()
Return the number of pixels of the height of the screen of this widget in mm.

wininfo_screennmwidth ()
Return the number of pixels of the width of the screen of this widget in mm.

wininfo_screensvisual ()
Return one of the strings directcolor, grayscale, pseudocolor, staticcolor, staticgray, or truecolor for the
default colormodel of this screen.

wininfo_screenwidth ()
Return the number of pixels of the width of the screen of this widget in pixel.

wininfo_server ()
Return information of the X-Server of the screen of this widget in the form “XmajorRminor vendor ven-
dorVersion”.

wininfo_toplevel ()
Return the toplevel widget of this widget.

wininfo_viewable ()
Return true if the widget and all its higher ancestors are mapped.

wininfo_visual ()
Return one of the strings directcolor, grayscale, pseudocolor, staticcolor, staticgray, or truecolor for the
colormodel of this widget.

wininfo_visualid ()
Return the X identifier for the visual for this widget.

wininfo_visualsavailable (includeids=0)
Return a list of all visuals available for the screen of this widget.

Each item in the list consists of a visual name (see wininfo_visual), a depth and if INCLUDEIDS=1 is given
also the X identifier.
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, XXXXXX
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”.

wm_protocol (name=None, func=None)

Bind function FUNC to command NAME for this widget. Return the function bound to NAME if None is given. NAME could be e.g. “WM_SAVE_YOURSELF” or “WM_DELETE_WINDOW”.
wm_resizable(width=None, height=None)
   Instruct the window manager whether this width can be resized in WIDTH or HEIGHT. Both values are
   boolean values.

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

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

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

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

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

class robot.libraries.dialogs_py.InputDialog(message, default='', hidden=False)
   Bases: robot.libraries.dialogs_py._TkDialog

after(ms, func=None, *args)
   Call function once after given time.
   MS specifies the time in milliseconds. FUNC gives the function which shall be called. Additional parameters are given as parameters to the function call. Return identifier to cancel scheduling with after_cancel.

after_cancel(id)
   Cancel scheduling of function identified with ID.
   Identifier returned by after or after_idle must be given as first parameter.

after_idle(func, *args)
   Call FUNC once if the Tcl main loop has no event to process.
   Return an identifier to cancel the scheduling with after_cancel.

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

attributes(*args)
   This subcommand returns or sets platform specific attributes
   The first form returns a list of the platform specific flags and their values. The second form returns the
   value for the specific option. The third form sets one or more of the values. The values are as follows:
   On Windows, -disabled gets or sets whether the window is in a disabled state. -toolwindow gets or sets
   the style of the window to toolwindow (as defined in the MSDN). -topmost gets or sets whether this is a
topmost window (displays above all other windows).
   On Macintosh, XXXXX
   On Unix, there are currently no special attribute values.

bbox(column=None, row=None, col2=None, row2=None)
   Return a tuple of integer coordinates for the bounding box of this widget controlled by the geometry
   manager grid.

4.1. robot package
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.
```plaintext
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.

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

deletecommand (name)
Internal function.

Delete the Tcl command provided in NAME.

destroy ()
Destroy this and all descendants widgets.

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

4.1. robot package 121
event_delete (virtual, sequences)
Unbind a virtual event VIRTUAL from SEQUENCE.

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

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

focus ()
Direct input focus to this widget.

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

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

Return None if the application does not have the focus.

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

focus_get ()
Return the widget which has currently the focus in the application.
Use focus_displayof to allow working with several displays. Return None if application does not have the focus.

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

focus_set ()
Direct input focus to this widget.

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

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

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

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

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

getdouble
alias of __builtin__.float

getint
alias of __builtin__.int

getvar (name='PY_VAR')
Return value of Tcl variable NAME.
grab_current ()
   Return widget which has currently the grab in this application or None.

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

grab_set (timeout=30)

grab_set_global ()
   Set global grab for this widget.
   A global grab directs all events to this and descendant widgets on the display. Use with caution - other
   applications do not get events anymore.

grab_status ()
   Return None, “local” or “global” if this widget has no, a local or a global grab.

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

grid_bbox (column=None, row=None, col2=None, row2=None)
   Return a tuple of integer coordinates for the bounding box of this widget controlled by the geometry
   manager grid.
   If COLUMN, ROW is given the bounding box applies from the cell with row and column 0 to the specified
   cell. If COL2 and ROW2 are given the bounding box starts at that cell.
   The returned integers specify the offset of the upper left corner in the master widget and the width and
   height.

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

grid_location (x, y)
   Return a tuple of column and row which identify the cell at which the pixel at position X and Y inside the
   master widget is located.

grid_propagate (flag=’_noarg_’)
   Set or get the status for propagation of geometry information.
   A boolean argument specifies whether the geometry information of the slaves will determine the size of
   this widget. If no argument is given, the current setting will be returned.

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

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

grid_slaves (row=None, column=None)
   Return a list of all slaves of this widget in its packing order.

group (pathName=None)
   Set the group leader widgets for related widgets to PATHNAME. Return the group leader of this widget if
   None is given.
iconbitmap (bitmap=None, default=None)
Set bitmap for the iconified widget to BITMAP. Return the bitmap if None is given.

Under Windows, the DEFAULT parameter can be used to set the icon for the widget and any 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 pa-
rameters 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_PhysicalRequest.

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

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

wm_iconify()
Display widget as icon.

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

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

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

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

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

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

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

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

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.

**class** robot.libraries.dialogs_py.SelectionDialog *(message, values)*

Bases: robot.libraries.dialogs_py._TkDialog

**after** *(ms, func=None, *args)*

Call function once after given time.

MS specifies the time in milliseconds. FUNC gives the function which shall be called. Additional parameters are given as parameters to the function call. Return identifier to cancel scheduling with after_cancel.

**after_cancel** *(id)*

Cancel scheduling of function identified with ID.

Identifier returned by after or after_idle must be given as first parameter.

**after_idle** *(func, *args)*

Call FUNC once if the Tcl main loop has no event to process.

Return an identifier to cancel the scheduling with after_cancel.

**aspect** *(minNumer=None, minDenom=None, maxNumer=None, maxDenom=None)*

Instruct the window manager to set the aspect ratio (width/height) of this widget to be between MINNUMER/MINDENOM and MAXNUMER/MAXDENOM. Return a tuple of the actual values if no argument is given.

**attributes** *(*args)*

This subcommand returns or sets platform specific attributes

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

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

On Macintosh, XXXXX

On Unix, there are currently no special attribute values.
bbox (column=None, row=None, col2=None, row2=None)
Return a tuple of integer coordinates for the bounding box of this widget controlled by the geometry manager grid.

If COLUMN, ROW is given the bounding box applies from the cell with row and column 0 to the specified cell. If COL2 and ROW2 are given the bounding box starts at that cell.

The returned integers specify the offset of the upper left corner in the master widget and the width and height.

bell (displayof=0)
Ring a display’s bell.

bind (sequence=None, func=None, add=None)
Bind to this widget at event SEQUENCE a call to function FUNC.

SEQUENCE is a string of concatenated event patterns. An event pattern is of the form <MODIFIER-MODIFIER-TYPE-DETAIL> where MODIFIER is one of Control, Mod2, M2, Shift, Mod3, M3, Lock, Mod4, M4, Button1, B1, Mod5, M5 Button2, B2, Meta, M, Button3, B3, Alt, Button4, B4, Double, Button5, B5 Triple, Mod1, M1. TYPE is one of Activate, Enter, Map, ButtonPress, Button, Expose, Motion, ButtonRelease FocusIn, MouseWheel, Circulate, FocusOut, Property, Colormap, Gravity Reparent, Configure, KeyPress, Key, Unmap, Deactivate, KeyRelease Visibility, Destroy, Leave and DETAIL is the button number for ButtonPress, ButtonRelease and DETAIL is the Keysym for KeyPress and KeyRelease. Examples are <Control-Button-1> for pressing Control and mouse button 1 or <Alt-A> for pressing A and the Alt key (KeyPress can be omitted). An event pattern can also be a virtual event of the form <<ASTRING>> where AString can be arbitrary. This event can be generated by event_generate. If events are concatenated they must appear shortly after each other.

FUNC will be called if the event sequence occurs with an instance of Event as argument. If the return value of FUNC is “break” no further bound function is invoked.

An additional boolean parameter ADD specifies whether FUNC will be called additionally to the other bound function or whether it will replace the previous function.

Bind will return an identifier to allow deletion of the bound function with unbind without memory leak.

If FUNC or SEQUENCE is omitted the bound function or list of bound events are returned.

bind_all (sequence=None, func=None, add=None)
Bind to all widgets at an event SEQUENCE a call to function FUNC. An additional boolean parameter ADD specifies whether FUNC will be called additionally to the other bound function or whether it will replace the previous function. See bind for the return value.

bind_class (className, sequence=None, func=None, add=None)
Bind to widgets with bindtag CLASSNAME at event SEQUENCE a call of function FUNC. An additional boolean parameter ADD specifies whether FUNC will be called additionally to the other bound function or whether it will replace the previous function. See bind for the return value.

bindtags (tagList=None)
Set or get the list of bindtags for this widget.

With no argument return the list of all bindtags associated with this widget. With a list of strings as argument the bindtags are set to this list. The bindtags determine in which order events are processed (see bind).

cget (key)
Return the resource value for a KEY given as string.

client (name=None)
Store NAME in WM_CLIENT_MACHINE property of this widget. Return current value.
**clipboard_append** *(string, **kw]*)

Append STRING to the Tk clipboard.

A widget specified at the optional displayof keyword argument specifies the target display. The clipboard can be retrieved with **selection_get**.

**clipboard_clear** (**kw**)

Clear the data in the Tk clipboard.

A widget specified for the optional displayof keyword argument specifies the target display.

**clipboard_get** (**kw**)

Retrieve data from the clipboard on window’s display.

The window keyword defaults to the root window of the Tkinter application.

The type keyword specifies the form in which the data is to be returned and should be an atom name such as STRING or FILE_NAME. Type defaults to STRING, except on X11, where the default is to try UTF8_STRING and fall back to STRING.

This command is equivalent to:

```
selection_get(CLIPBOARD)
```

**colormapwindows** (*wlist*)

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

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

Useless. Not implemented in Tk.

**columnconfigure** *(index, cnf=[], **kw]*)

Configure column INDEX of a grid.

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

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

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

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

Configure resources of a widget.

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

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

Configure resources of a widget.

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

**deiconify** ()

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

**deletecommand** *(name)*

Internal function.

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

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

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

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

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

**focus** ()
Direct input focus to this widget.

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

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

    Return None if the application does not have the focus.

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

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

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

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

**focus_set** ()
Direct input focus to this widget.

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

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

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

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

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

**getdouble**
alias of __builtin__.float
getint
   alias of __builtin__.int

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

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

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

grab_set (timeout=30)

grab_set_global ()
   Set global grab for this widget.
   A global grab directs all events to this and descendant widgets on the display. Use with caution - other applications do not get events anymore.

grab_status ()
   Return None, “local” or “global” if this widget has no, a local or a global grab.

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

grid_bbox (column=None, row=None, col2=None, row2=None)
   Return a tuple of integer coordinates for the bounding box of this widget controlled by the geometry manager grid.
   If COLUMN, ROW is given the bounding box applies from the cell with row and column 0 to the specified cell. If COL2 and ROW2 are given the bounding box starts at that cell.
   The returned integers specify the offset of the upper left corner in the master widget and the width and height.

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

grid_location (x, y)
   Return a tuple of column and row which identify the cell at which the pixel at position X and Y inside the master widget is located.

grid_propagate (flag='*_noarg_*')
   Set or get the status for propagation of geometry information.
   A boolean argument specifies whether the geometry information of the slaves will determine the size of this widget. If no argument is given, the current setting will be returned.

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

grid_size ()
   Return a tuple of the number of column and rows in the grid.
**grid_slaves** *(row=None, column=None)*
Return a list of all slaves of this widget in its packing order.

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

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

Under Windows, the DEFAULT parameter can be used to set the icon for the widget and any 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.

---

4.1. robot package 139
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)}
Return integer which represents atom NAME.

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

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

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

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

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

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

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

\texttt{winfo\_exists()}
Return true if this widget exists.

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

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

\texttt{winfo\_height()}
Return height of this widget.

\texttt{winfo\_id()}
Return identifier ID for this widget.

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

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

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

\texttt{winfo\_name()}
Return the name of this widget.

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

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

\texttt{winfo\_pixels(number)}
Rounded integer value of \texttt{winfo\_fpixels}.

\texttt{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.
\textbf{winfo\_visualsavailable} \texttt{(includeids=0)}

Return a list of all visuals available for the screen of this widget.

Each item in the list consists of a visual name (see \texttt{winfo\_visual}), a depth and if INCLUDEIDS=1 is given also the X identifier.

\textbf{winfo\_vrootheight}\texttt{()}

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.

\textbf{winfo\_vrootwidth}\texttt{()}

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.

\textbf{winfo\_vrootx}\texttt{()}

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

\textbf{winfo\_vrooty}\texttt{()}

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

\textbf{winfo\_width}\texttt{()}

Return the width of this widget.

\textbf{winfo\_x}\texttt{()}

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

\textbf{winfo\_y}\texttt{()}

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

\textbf{withdraw}\texttt{()}

Withdraw this widget from the screen such that it is unmapped and forgotten by the window manager. Re-draw it with \texttt{wm\_deiconify}.

\textbf{wm\_aspect}\texttt{(minNumer=None, minDenom=None, maxNumer=None, maxDenom=None)}

Instruct the window manager to set the aspect ratio (width/height) of this widget to be between MINNUMER/MINDENOM and MAXNUMER/MAXDENOM. Return a tuple of the actual values if no argument is given.

\textbf{wm\_attributes}\texttt{(*args)}

This subcommand returns or sets platform specific attributes

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

On Windows, \texttt{-disabled} gets or sets whether the window is in a disabled state. \texttt{-toolwindow} gets or sets the style of the window to toolwindow (as defined in the MSDN). \texttt{-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.

\textbf{wm\_client}\texttt{(name=None)}

Store \texttt{NAME} in \texttt{WM\_CLIENT\_MACHINE} property of this widget. Return current value.

\textbf{wm\_colormapwindows}\texttt{(*wlist)}

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

\textbf{wm\_command}\texttt{(value=None)}

Store \texttt{VALUE} in \texttt{WM\_COMMAND} property. It is the command which shall be used to invoke the application. Return current command if \texttt{VALUE} is None.

4.1. robot package
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”.

Chapter 4. All packages
wm_protocol(name=None, func=None)
Bind function FUNC to command NAME for this widget. Return the function bound to NAME if None is given. NAME could be e.g. “WM_SAVE_YOURSELF” or “WM_DELETE_WINDOW”.

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

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

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

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

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

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

class robot.libraries.dialogs_py.MultipleSelectionDialog(message, values)
Bases: robot.libraries.dialogs_py._TkDialog

after(ms, func=None, *args)
Call function once after given time.

MS specifies the time in milliseconds. FUNC gives the function which shall be called. Additional parameters are given as parameters to the function call. Return identifier to cancel scheduling with after_cancel.

after_cancel(id)
Cancel scheduling of function identified with ID.
Identifier returned by after or after_idle must be given as first parameter.

after_idle(func, *args)
Call FUNC once if the Tcl main loop has no event to process.

Return an identifier to cancel the scheduling with after_cancel.

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

attributes(*args)
This subcommand returns or sets platform specific attributes
The first form returns a list of the platform specific flags and their values. The second form returns the value for the specific option. The third form sets one or more of the values. The values are as follows:
On Windows, -disabled gets or sets whether the window is in a disabled state. -toolwindow gets or sets the style of the window to toolwindow (as defined in the MSDN). -topmost gets or sets whether this is a toplevel 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 <<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:

```plaintext
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 descendant widgets.

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

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

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

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

**focus()**
Direct input focus to this widget.

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

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

Return None if the application does not have the focus.

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

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

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

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

**focus_set()**
Direct input focus to this widget.

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

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

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

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

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

**getdouble**
alias of __builtin__.float
getint
   alias of __builtin__.int

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

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

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

grab_set (timeout=30)

grab_set_global()
   Set global grab for this widget.
   A global grab directs all events to this and descendant widgets on the display. Use with caution - other applications do not get events anymore.

grab_status()
   Return None, “local” or “global” if this widget has no, a local or a global grab.

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

grid_bbox (column=None, row=None, col2=None, row2=None)
   Return a tuple of integer coordinates for the bounding box of this widget controlled by the geometry manager grid.
   If COLUMN, ROW is given the bounding box applies from the cell with row and column 0 to the specified cell. If COL2 and ROW2 are given the bounding box starts at that cell.
   The returned integers specify the offset of the upper left corner in the master widget and the width and height.

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

grid_location (x, y)
   Return a tuple of column and row which identify the cell at which the pixel at position X and Y inside the master widget is located.

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

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

grid_size()
   Return a tuple of the number of column and rows in the grid.
**grid_slaves** (*row=None, column=None*)

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

**group** (*pathName=None*)

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

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

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

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

**iconify**

Display widget as icon.

**iconmask** (*bitmap=None*)

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

**iconname** (*newName=None*)

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

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

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

**iconwindow** (*pathName=None*)

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

**image_names**

Return a list of all existing image names.

**image_types**

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

**keys**

Return a list of all resource names of this widget.

**lift** (*aboveThis=None*)

Raise this widget in the stacking order.

**lower** (*belowThis=None*)

Lower this widget in the stacking order.

**mainloop** (*n=0*)

Call the mainloop of Tk.

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

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

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

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

**nametowidget** (*name*)

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

**option_add** (*pattern, value, priority=None*)

Set a VALUE (second parameter) for an option PATTERN (first parameter).

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Return the contents of the current X selection.

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

**selection_handle**(command, **kw**)

Specify a function COMMAND to call if the X selection owned by this widget is queried by another application.

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

**selection_own**(**kw**)

Become owner of X selection.

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

**selection_own_get**(**kw**)

Return owner of X selection.

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

**send**(interp, cmd, *args**)

Send Tcl command CMD to different interpreter INTERP to be executed.

**setvar**(name='PY_VAR', value='1')

Set Tcl variable NAME to VALUE.

**show**( )

**size**( )

Return a tuple of the number of column and rows in the grid.

**sizefrom**(who=None)

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

**slaves**( )

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

**state**(newstate=None)

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

**title**(string=None)

Set the title of this widget.

**tk_bisque**( )

Change the color scheme to light brown as used in Tk 3.6 and before.

**tk_focusFollowsMouse**( )

The widget under mouse will get automatically focus. Can not be disabled easily.

**tk_focusNext**( )

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

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

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

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

\begin{itemize}
\item 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: \texttt{activeBackground}, \texttt{foreground}, \texttt{selectColor}, \texttt{activeForeground}, \texttt{highlightBackground}, \texttt{selectBackground}, \texttt{background}, \texttt{highlightColor}, \texttt{selectForeground}, \texttt{disabledForeground}, \texttt{insertBackground}, \texttt{troughColor}.
\end{itemize}

\texttt{tk\_strictMotif(\texttt{boolean}=\texttt{None})}  
Set Tcl internal variable, whether the look and feel should adhere to Motif.

\begin{itemize}
\item A parameter of \texttt{1} means adhere to Motif (e.g. no color change if mouse passes over slider). Returns the set value.
\end{itemize}

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

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

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

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

\texttt{unbind\_class(\texttt{className, sequence})}  
Unbind for all widgets with bindtag \texttt{CLASSNAME} for event \texttt{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(\texttt{name}=’PY\_VAR’)\textbackslash name}  
Wait until the variable is modified.

\begin{itemize}
\item A parameter of type \texttt{IntVar}, \texttt{StringVar}, \texttt{DoubleVar} or \texttt{BooleanVar} must be given.
\end{itemize}

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

\begin{itemize}
\item If no parameter is given self is used.
\end{itemize}

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

\begin{itemize}
\item If no parameter is given self is used.
\end{itemize}

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

```python
winfo_atom(name, displayof=0)
Return integer which represents atom NAME.
```

```python
winfo_atomname(id, displayof=0)
Return name of atom with identifier ID.
```

```python
winfo_cells()
Return number of cells in the colormap for this widget.
```

```python
winfo_children()
Return a list of all widgets which are children of this widget.
```

```python
winfo_class()
Return window class name of this widget.
```

```python
winfo_colormapfull()
Return true if at the last color request the colormap was full.
```

```python
winfo_containing(rootX, rootY, displayof=0)
Return the widget which is at the root coordinates ROOTX, ROOTY.
```

```python
winfo_depth()
Return the number of bits per pixel.
```

```python
winfo_exists()
Return true if this widget exists.
```

```python
winfo_fpixels(number)
Return the number of pixels for the given distance NUMBER (e.g. “3c”) as float.
```

```python
winfo_geometry()
Return geometry string for this widget in the form “widthxheight+X+Y”.
```

```python
winfo_height()
Return height of this widget.
```

```python
winfo_id()
Return identifier ID for this widget.
```

```python
winfo_interps(displayof=0)
Return the name of all Tcl interpreters for this display.
```

```python
winfo_ismapped()
Return true if this widget is mapped.
```

```python
winfo_manager()
Return the window manager name for this widget.
```

```python
winfo_name()
Return the name of this widget.
```

```python
winfo_parent()
Return the name of the parent of this widget.
```

```python
winfo_pathname(id, displayof=0)
Return the pathname of the widget given by ID.
```

```python
winfo_pixels(number)
Rounded integer value of winfo_fpixels.
```

```python
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_pointerxy()
    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 ven-
dorVersion”.

winfo_toplevel()
    Return the toplevel widget of this widget.

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

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

winfo_visualid()
    Return the X identifier for the visual for this widget.
winfo_visualsavailable (includeids=0)
Return a list of all visuals available for the screen of this widget.
Each item in the list consists of a visual name (see winfo_visual), a depth and if INCLUDEIDS=1 is given also the X identifier.

winfo_vrootheight()
Return the height of the virtual root window associated with this widget in pixels. If there is no virtual root window return the height of the screen.

winfo_vrootwidth()
Return the width of the virtual root window associated with this widget in pixel. If there is no virtual root window return the width of the screen.

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

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

winfo_width()
Return the width of this widget.

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

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

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

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

wm_attributes (*args)
This subcommand returns or sets platform specific attributes
The first form returns a list of the platform specific flags and their values. The second form returns the value for the specific option. The third form sets one or more of the values. The values are as follows:
On Windows, -disabled gets or sets whether the window is in a disabled state. -toolwindow gets or sets the style of the window to toolwindow (as defined in the MSDN). -topmost gets or sets whether this is a topmost window (displays above all other windows).
On Macintosh, XXXXX
On Unix, there are currently no special attribute values.

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

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

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

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

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

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

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

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

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

wm_iconify()
Display widget as icon.

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

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

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

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

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

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

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

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

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.

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:

```python
selection_get(CLIPBOARD)
```

**colormapwindows** *( *wlist)*

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

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

Useless. Not implemented in Tk.

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

Configure column INDEX of a grid.

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

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

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

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

Configure resources of a widget.

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

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

Configure resources of a widget.

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

**deiconify** ()

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

**deletecommand** *(name)*

Internal function.

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

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

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

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

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

focus()
  Direct input focus to this widget.
  If the application currently does not have the focus this widget will get the focus if the application gets the focus through the window manager.

focus_displayof()
  Return the widget which has currently the focus on the display where this widget is located.
  Return None if the application does not have the focus.

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

focus_get()
  Return the widget which has currently the focus in the application.
  Use focus_displayof to allow working with several displays. Return None if application does not have the focus.

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

focus_set()
  Direct input focus to this widget.
  If the application currently does not have the focus this widget will get the focus if the application gets the focus through the window manager.

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

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

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

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

getdouble
  alias of __builtin__.float

4.1. robot package
getint
   alias of __builtin__.int

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

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

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

grab_set (timeout=30)

grab_set_global ()
   Set global grab for this widget.

   A global grab directs all events to this and descendant widgets on the display. Use with caution - other
   applications do not get events anymore.

grab_status ()
   Return None, “local” or “global” if this widget has no, a local or a global grab.

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

grid_bbox (column=None, row=None, col2=None, row2=None)
   Return a tuple of integer coordinates for the bounding box of this widget controlled by the geometry
   manager grid.

   If COLUMN, ROW is given the bounding box applies from the cell with row and column 0 to the specified
   cell. If COL2 and ROW2 are given the bounding box starts at that cell.

   The returned integers specify the offset of the upper left corner in the master widget and the width and
   height.

grid_columnconfigure (index, cnf={}, **kw)
   Configure column INDEX of a grid.

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

grid_location (x, y)
   Return a tuple of column and row which identify the cell at which the pixel at position X and Y inside the
   master widget is located.

grid_propagate (flag=['_noarg_'])
   Set or get the status for propagation of geometry information.

   A boolean argument specifies whether the geometry information of the slaves will determine the size of
   this widget. If no argument is given, the current setting will be returned.

grid_rowconfigure (index, cnf={}, **kw)
   Configure row INDEX of a grid.

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

grid_size ()
   Return a tuple of the number of column and rows in the grid.
grid_slaves (row=None, column=None)
Return a list of all slaves of this widget in its packing order.

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

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

iconify ()
Display widget as icon.

iconmask (bitmap=None)
Set mask for the icon bitmap of this widget. Return the mask if None is given.

iconname (newName=None)
Set the name of the icon for this widget. Return the name if None is given.

iconposition (x=None, y=None)
Set the position of the icon of this widget to X and Y. Return a tuple of the current values of X and X if
None is given.

iconwindow (pathName=None)
Set widget PATHNAME to be displayed instead of icon. Return the current value if None is given.

image_names ()
Return a list of all existing image names.

image_types ()
Return a list of all available image types (e.g. photo bitmap).

keys ()
Return a list of all resource names of this widget.

lift (aboveThis=None)
Raise this widget in the stacking order.

lower (belowThis=None)
Lower this widget in the stacking order.

mainloop (n=0)
Call the mainloop of Tk.

maxsize (width=None, height=None)
Set max WIDTH and HEIGHT for this widget. If the window is gridded the values are given in grid units.
Return the current values if None is given.

minsize (width=None, height=None)
Set min WIDTH and HEIGHT for this widget. If the window is gridded the values are given in grid units.
Return the current values if None is given.

nametowidget (name)
Return the Tkinter instance of a widget identified by its Tcl name NAME.

option_add (pattern, value, priority=None)
Set a VALUE (second parameter) for an option PATTERN (first parameter).
An optional third parameter gives the numeric priority (defaults to 80).
option_clear()
    Clear the option database.
    It will be reloaded if option_add is called.

option_get(name, className)
    Return the value for an option NAME for this widget with CLASSNAME.
    Values with higher priority override lower values.

option_readfile(fileName, priority=None)
    Read file FILENAME into the option database.
    An optional second parameter gives the numeric priority.

overrideredirect(boolean=None)
    Instruct the window manager to ignore this widget if BOOLEAN is given with 1. Return the current value
    if None is given.

pack_propagate(flag=['_noarg_'])
    Set or get the status for propagation of geometry information.
    A boolean argument specifies whether the geometry information of the slaves will determine the size of
    this widget. If no argument is given the current setting will be returned.

pack_slaves()
    Return a list of all slaves of this widget in its packing order.

place_slaves()
    Return a list of all slaves of this widget in its packing order.

positionfrom(who=None)
    Instruct the window manager that the position of this widget shall be defined by the user if WHO is "user",
    and by its own policy if WHO is "program".

propagate(flag=['_noarg_'])
    Set or get the status for propagation of geometry information.
    A boolean argument specifies whether the geometry information of the slaves will determine the size of
    this widget. If no argument is given the current setting will be returned.

protocol(name=None, func=None)
    Bind function FUNC to command NAME for this widget. Return the function bound to NAME if None is
    given. NAME could be e.g. "WM_SAVE_YOURSELF" or "WM_DELETE_WINDOW".

quit()
    Quit the Tcl interpreter. All widgets will be destroyed.

register(func, subst=None, needcleanup=1)
    Return a newly created Tcl function. If this function is called, the Python function FUNC will be executed.
    An optional function SUBST can be given which will be executed before FUNC.

resizable(width=None, height=None)
    Instruct the window manager whether this width can be resized in WIDTH or HEIGHT. Both values are
    boolean values.

rowconfigure(index, cnf={}, **kw)
    Configure row INDEX of a grid.
    Valid resources are minsize (minimum size of the row), weight (how much does additional space propagate
to this row) and pad (how much space to let additionally).

selection_clear(**kw)
    Clear the current X selection.
selection_get(**kw)

Return the contents of the current X selection.

A keyword parameter selection specifies the name of the selection and defaults to PRIMARY. A keyword parameter displayof specifies a widget on the display to use. A keyword parameter type specifies the form of data to be fetched, defaulting to STRING except on X11, where UTF8_STRING is tried before STRING.

selection_handle(command, **kw)

Specify a function COMMAND to call if the X selection owned by this widget is queried by another application.

This function must return the contents of the selection. The function will be called with the arguments OFFSET and LENGTH which allows the chunking of very long selections. The following keyword 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}(*\texttt{args}) 
Do not use. Needed in Tk 3.6 and earlier.

\texttt{tk\_setPalette}(*\texttt{args}, **\texttt{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, highlightBackgroundColor, selectBackgroundColor, background, highlightColor, selectForeground, disabledForeground, insertBackgroundColor, troughColor.

\texttt{tk\_strictMotif}(boolean=None) 
Set Tcl internal variable, whether the look and feel should adhere to Motif.

A parameter of \texttt{1} means adhere to Motif (e.g. no color change if mouse passes over slider). Returns the set value.

\texttt{tkraise}(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 \texttt{SEQUENCE} the function identified with \texttt{FUNCID}.

\texttt{unbind\_all}(sequence) 
Unbind for all widgets for event \texttt{SEQUENCE} all functions.

\texttt{unbind\_class}(className, sequence) 
Unbind for all widgets with bindtag \texttt{CLASSNAME} for event \texttt{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='\texttt{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{waitvar}(name='\texttt{PY\_VAR}') 
Wait until the variable is modified.
A parameter of type IntVar, StringVar, DoubleVar or BooleanVar must be given.

```python
winfo_atom(name, displayof=0)
Return integer which represents atom NAME.
```

```python
winfo_atomname(id, displayof=0)
Return name of atom with identifier ID.
```

```python
winfo_cells()
Return number of cells in the colormap for this widget.
```

```python
winfo_children()
Return a list of all widgets which are children of this widget.
```

```python
winfo_class()
Return window class name of this widget.
```

```python
winfo_colormapfull()
Return true if at the last color request the colormap was full.
```

```python
winfo_containing(rootX, rootY, displayof=0)
Return the widget which is at the root coordinates ROOTX, ROOTY.
```

```python
winfo_depth()
Return the number of bits per pixel.
```

```python
winfo_exists()
Return true if this widget exists.
```

```python
winfo_fpixels(number)
Return the number of pixels for the given distance NUMBER (e.g. “3c”) as float.
```

```python
winfo_geometry()
Return geometry string for this widget in the form “widthxheight+X+Y”.
```

```python
winfo_height()
Return height of this widget.
```

```python
winfo_id()
Return identifier ID for this widget.
```

```python
winfo_interps(displayof=0)
Return the name of all Tcl interpreters for this display.
```

```python
winfo_ismapped()
Return true if this widget is mapped.
```

```python
winfo_manager()
Return the window manager name for this widget.
```

```python
winfo_name()
Return the name of this widget.
```

```python
winfo_parent()
Return the name of the parent of this widget.
```

```python
winfo_pathname(id, displayof=0)
Return the pathname of the widget given by ID.
```

```python
winfo_pixels(number)
Rounded integer value of winfo_fpixels.
```

```python
winfo_pointerx()
Return the x coordinate of the pointer on the root window.
```
winfo_pointerx()  
    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 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 X if None is given.

wm_iconwindow(pathName=None)
   Set widget PATHNAME to be displayed instead of icon. Return the current value if None is given.

wm_maxsize(width=None, height=None)
   Set max WIDTH and HEIGHT for this widget. If the window is gridded the values are given in grid units. Return the current values if None is given.

wm_minsize(width=None, height=None)
   Set min WIDTH and HEIGHT for this widget. If the window is gridded the values are given in grid units. Return the current values if None is given.

wm_overrideredirect(boolean=None)
   Instruct the window manager to ignore this widget if BOOLEAN is given with 1. Return the current value if None is given.

wm_positionfrom(who=None)
   Instruct the window manager that the position of this widget shall be defined by the user if WHO is “user”, and by its own policy if WHO is “program”.

**wm_protocol** *(name=None, func=None)*

Bind function FUNC to command NAME for this widget. Return the function bound to NAME if None is given. NAME could be e.g. “WM_SAVE_YOURSELF” or “WM_DELETE_WINDOW”.

**wm_resizable** *(width=None, height=None)*

Instruct the window manager whether this width can be resized in WIDTH or HEIGHT. Both values are boolean values.

**wm_sizefrom** *(who=None)*

Instruct the window manager that the size of this widget shall be defined by the user if WHO is “user”, and by its own policy if WHO is “program”.

**wm_state** *(newstate=None)*

Query or set the state of this widget as one of normal, icon, iconic (see wm_iconwindow), withdrawn, or zoomed (Windows only).

**wm_title** *(string=None)*

Set the title of this widget.

**wm_transient** *(master=None)*

Instruct the window manager that this widget is transient with regard to widget MASTER.

**wm_withdraw** *

Withdraw this widget from the screen such that it is unmapped and forgotten by the window manager. Re-draw it with wm_deiconify.

---

**robot.model package**

Package with generic, reusable and extensible model classes.

This package contains, for example, TestSuite, TestCase, Keyword and SuiteVisitor base classes. These classes are extended both by execution and result related model objects and used also elsewhere.

This package is considered stable.

---

**Submodules**

**robot.model.configurer module**

**class** robot.model.configurer.SuiteConfigurer *(name=None, doc=None, metadata=None, set_tags=None, include_tags=None, exclude_tags=None, include_suites=None, include_tests=None, empty_suite_ok=False)*

**Bases:** robot.model.visitor.SuiteVisitor

**add_tags**

**remove_tags**

**visit_suite** *(suite)*

Implements traversing through the suite and its direct children.

Can be overridden to allow modifying the passed in suite without calling start_suite() or end_suite() nor visiting child suites, tests or keywords (setup and teardown) at all.

**end_keyword** *(keyword)*

Called when keyword ends. Default implementation does nothing.
**end_message** *(msg)*
Called when message ends. Default implementation does nothing.

**end_suite** *(suite)*
Called when suite ends. Default implementation does nothing.

**end_test** *(test)*
Called when test ends. Default implementation does nothing.

**start_keyword** *(keyword)*
Called when keyword starts. Default implementation does nothing.
Can return explicit False to stop visiting.

**start_message** *(msg)*
Called when message starts. Default implementation does nothing.
Can return explicit False to stop visiting.

**start_suite** *(suite)*
Called when suite starts. Default implementation does nothing.
Can return explicit False to stop visiting.

**start_test** *(test)*
Called when test starts. Default implementation does nothing.
Can return explicit False to stop visiting.

**visit_keyword** *(kw)*
Implements traversing through the keyword and its child keywords.
Can be overridden to allow modifying the passed in kw without calling start_keyword() or end_keyword() nor visiting child keywords.

**visit_message** *(msg)*
Implements visiting the message.
Can be overridden to allow modifying the passed in msg without calling start_message() or end_message().

**visit_test** *(test)*
Implements traversing through the test and its keywords.
Can be overridden to allow modifying the passed in test without calling start_test() or end_test() nor visiting keywords.

---

**robot.model.criticality module**

**class** robot.model.criticality.Criticality *(critical_tags=None, non_critical_tags=None)*
Bases: object

- **tag_is_critical** *(tag)*
- **tag_is_non_critical** *(tag)*
- **test_is_critical** *(test)*

---

**robot.model.filter module**

**class** robot.model.filter.EmptySuiteRemover *(preserve_direct_children=False)*
Bases: robot.model.visitor.SuiteVisitor
end_suite(suite)
   Called when suite ends. Default implementation does nothing.

visit_test(test)
   Implements traversing through the test and its keywords.

   Can be overridden to allow modifying the passed in test without calling start_test() or end_test() nor visiting keywords.

visit_keyword(kw)
   Implements traversing through the keyword and its child keywords.

   Can be overridden to allow modifying the passed in kw without calling start_keyword() or end_keyword() nor visiting child keywords.

end_keyword(keyword)
   Called when keyword ends. Default implementation does nothing.

end_message(msg)
   Called when message ends. Default implementation does nothing.

end_test(test)
   Called when test ends. Default implementation does nothing.

start_keyword(keyword)
   Called when keyword starts. Default implementation does nothing.

   Can return explicit False to stop visiting.

start_message(msg)
   Called when message starts. Default implementation does nothing.

   Can return explicit False to stop visiting.

start_suite(suite)
   Called when suite starts. Default implementation does nothing.

   Can return explicit False to stop visiting.

start_test(test)
   Called when test starts. Default implementation does nothing.

   Can return explicit False to stop visiting.

visit_message(msg)
   Implements visiting the message.

   Can be overridden to allow modifying the passed in msg without calling start_message() or end_message().

visit_suite(suite)
   Implements traversing through the suite and its direct children.

   Can be overridden to allow modifying the passed in suite without calling start_suite() or end_suite() nor visiting child suites, tests or keywords (setup and teardown) at all.

class robot.model.filter.Filter(include_suites=None, include_tests=None, include_tags=None, exclude_tags=None)
   Bases: robot.model.filter.EmptySuiteRemover

   include_suites
   include_tests
   include_tags
exclude_tags

start_suite(suite)
  Called when suite starts. Default implementation does nothing.
  Can return explicit False to stop visiting.

date_keyword(keyword)
  Called when keyword ends. Default implementation does nothing.

date_message(msg)
  Called when message ends. Default implementation does nothing.

date_suite(suite)
  Called when suite ends. Default implementation does nothing.

date_test(test)
  Called when test ends. Default implementation does nothing.

start_keyword(keyword)
  Called when keyword starts. Default implementation does nothing.
  Can return explicit False to stop visiting.

start_message(msg)
  Called when message starts. Default implementation does nothing.
  Can return explicit False to stop visiting.

start_test(test)
  Called when test starts. Default implementation does nothing.
  Can return explicit False to stop visiting.

visit_keyword(kw)
  Implements traversing through the keyword and its child keywords.
  Can be overridden to allow modifying the passed in kw without calling start_keyword() or end_keyword() nor visiting child keywords.

visit_message(msg)
  Implements visiting the message.
  Can be overridden to allow modifying the passed in msg without calling start_message() or end_message().

visit_suite(suite)
  Implements traversing through the suite and its direct children.
  Can be overridden to allow modifying the passed in suite without calling start_suite() or end_suite() nor visiting child suites, tests or keywords (setup and teardown) at all.

visit_test(test)
  Implements traversing through the test and its keywords.
  Can be overridden to allow modifying the passed in test without calling start_test() or end_test() nor visiting keywords.

robot.model.imports module
robot.model.itemlist module

```python
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)
    remove(item)
    index(item, *start_and_end)
    clear()
    visit(visitor)
    count(item)
    sort()
    reverse()
```

robot.model.keyword module

```python
class robot.model.keyword.Keyword(name='', doc='', args=(), assign=(), tags=(), timeout=None, type='kw',
                                 KEYWORD_TYPE = 'kw'
                                 SETUP_TYPE = 'setup'
                                 TEARDOWN_TYPE = 'teardown'
                                 FOR_LOOP_TYPE = 'for'
                                 FOR_ITEM_TYPE = 'foritem'
                                 keyword_class = None
                                 message_class
                                 alias of robot.model.message.Message
                                 doc
                                 args
                                 Keyword arguments as a list of strings.
```
assign
    Assigned variables as a list of strings.

timeout

type
    Keyword type as a string. The value is either KEYWORD_TYPE, SETUP_TYPE, TEARDOWN_TYPE,
    FOR_LOOP_TYPE or FOR_ITEM_TYPE constant defined on the class level.

name

parent
    Parent test suite, test case or keyword.

tags
    Keyword tags as a Tags object.

keywords
    Child keywords as a Keywords object.

messages
    Messages as a Messages object.

children
    Child keywords and messages in creation order.

id
    Keyword id in format like s1-t3-k1.
    See TestSuite.id for more information.

source

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 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.

depdeepcopy(**attributes)
    Return deep copy of this object.

    Parameters attributes – Attributes to be set for the returned copy automatically. For ex-
    ample, 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)

    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.
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.

append(item)
clear()
count(item)
create(*args, **kwargs)
create_setup(*args, **kwargs)
extend(items)
index(item, *start_and_end)
insert(index, item)
pop(*index)
remove(item)
reverse()
sort()
visit(visitor)

tear down
  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.
create_teardown(*args, **kwargs)
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 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.

deepecopy (**attributes)
   Return deep copy of this object.

   Parameters attributes – Attributes to be set for the returned copy automatically. For ex-
   ample, 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.

class robot.model.message.Messages (message_class=<class 'robot.model.message.Message'>, parent=None, messages=None)

Bases: robot.model.itemlist.ItemList

append (item)

clear ()

count (item)

create (*args, **kwargs)

extend (items)

index (item, *start_and_end)

insert (index, item)

pop (*index)

remove (item)

reverse ()

sort ()

visit (visitor)
robot.model.metadata module

class robot.model.metadata.Metadata(initial=None):
    Bases: robot.utils.normalizing.NormalizedDict

    clear() → None. Remove all items from D.
    copy()
    get(k[, d]) → D[k] if k in D, else d. d defaults to None.
    items() → list of D’s (key, value) pairs, as 2-tuples
    iteritems() → an iterator over the (key, value) items of D
    iterkeys() → an iterator over the keys of D
    itervalues() → an iterator over the values of D
    keys() → list of D’s keys
    pop(k[, d]) → v, remove specified key and return the corresponding value. If key is not found, d is returned if given, otherwise KeyError is raised.
    popitem() → (k, v), remove and return some (key, value) pair as a 2-tuple; but raise KeyError if D is empty.
    setdefault(k[, d]) → D.get(k,d), also set D[k]=d if k not in D
    update(E[, **F]) → None. Update D from mapping/iterable E and F. If E present and has a .keys() method, does: for k in E: D[k]=E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k]=v In either case, this is followed by: for k, v in F.items(): D[k]=v
    values() → list of D’s values

robot.model.modelobject module

class robot.model.modelobject.ModelObject:
    Bases: object

    copy(**attributes)
        Return shallow copy of this object.

        Parameters attributes – Attributes to be set for the returned copy automatically. For example, test.copy(name='New name').

        See also deepcopy(). The difference between these two is the same as with the standard copy.copy and copy.deepcopy functions that these methods also use internally.

        New in Robot Framework 3.0.1.

    deepcopy(**attributes)
        Return deep copy of this object.

        Parameters attributes – Attributes to be set for the returned copy automatically. For example, test.deepcopy(name='New name').

        See also copy(). The difference between these two is the same as with the standard copy.copy and copy.deepcopy functions that these methods also use internally.

        New in Robot Framework 3.0.1.
**robot.model.modifier module**

```python
class robot.model.modifier.ModelModifier(visitors, empty_suite_ok, logger)
Bases: robot.model.visitor.SuiteVisitor

visit_suite(suite)
    Implements traversing through the suite and its direct children.
    Can be overridden to allow modifying the passed in suite without calling start_suite() or end_suite() nor visiting child suites, tests or keywords (setup and teardown) at all.

disable_keyword(keyword)
    Called when keyword ends. Default implementation does nothing.

disable_message(msg)
    Called when message ends. Default implementation does nothing.

disable_suite(suite)
    Called when suite ends. Default implementation does nothing.

disable_test(test)
    Called when test ends. Default implementation does nothing.

start_keyword(keyword)
    Called when keyword starts. Default implementation does nothing.
    Can return explicit False to stop visiting.

start_message(msg)
    Called when message starts. Default implementation does nothing.
    Can return explicit False to stop visiting.

start_suite(suite)
    Called when suite starts. Default implementation does nothing.
    Can return explicit False to stop visiting.

start_test(test)
    Called when test starts. Default implementation does nothing.
    Can return explicit False to stop visiting.

visit_keyword(kw)
    Implements traversing through the keyword and its child keywords.
    Can be overridden to allow modifying the passed in kw without calling start_keyword() or end_keyword() nor visiting child keywords.

visit_message(msg)
    Implements visiting the message.
    Can be overridden to allow modifying the passed in msg without calling start_message() or end_message().

visit_test(test)
    Implements traversing through the test and its keywords.
    Can be overridden to allow modifying the passed in test without calling start_test() or end_test() nor visiting keywords.
```
robot.model.namepatterns module

```python
class robot.model.namepatterns.SuiteNamePatterns(patterns=None):
    Bases: robot.model.namepatterns._NamePatterns
    match(name, longname=None)

class robot.model.namepatterns.TestNamePatterns(patterns=None):
    Bases: robot.model.namepatterns._NamePatterns
    match(name, longname=None)
```

robot.model.statistics module

```python
class robot.model.statistics.Statistics(suite, suite_stat_level=-1, tag_stat_include=None, tag_stat_exclude=None, tag_stat_combine=None, tag_stat_link=None, rpa=False):
    Bases: object
    Container for total, suite and tag statistics.
    Accepted parameters have the same semantics as the matching command line options.
    total = None
        Instance of TotalStatistics.
    suite = None
        Instance of SuiteStatistics.
    tags = None
        Instance of TagStatistics.
    visit(visitor)

class robot.model.statistics.StatisticsBuilder(total_builder, suite_builder, tag_builder):
    Bases: robot.model.visitor.SuiteVisitor
    start_suite(suite)
        Called when suite starts. Default implementation does nothing.
        Can return explicit False to stop visiting.
    end_suite(suite)
        Called when suite ends. Default implementation does nothing.
    visit_test(test)
        Implements traversing through the test and its keywords.
        Can be overridden to allow modifying the passed in test without calling start_test() or end_test() nor visiting keywords.
    visit_keyword(kw)
        Implements traversing through the keyword and its child keywords.
        Can be overridden to allow modifying the passed in kw without calling start_keyword() or end_keyword() nor visiting child keywords.
    end_keyword(keyword)
        Called when keyword ends. Default implementation does nothing.
```
**end_message** *(msg)*
Called when message ends. Default implementation does nothing.

**end_test** *(test)*
Called when test ends. Default implementation does nothing.

**start_keyword** *(keyword)*
Called when keyword starts. Default implementation does nothing.
Can return explicit `False` to stop visiting.

**start_message** *(msg)*
Called when message starts. Default implementation does nothing.
Can return explicit `False` to stop visiting.

**start_test** *(test)*
Called when test starts. Default implementation does nothing.
Can return explicit `False` to stop visiting.

**visit_message** *(msg)*
Implements visiting the message.
Can be overridden to allow modifying the passed in `msg` without calling `start_message()` or `end_message()`.

**visit_suite** *(suite)*
Implements traversing through the suite and its direct children.
Can be overridden to allow modifying the passed in `suite` without calling `start_suite()` or `end_suite()` nor visiting child suites, tests or keywords (setup and teardown) at all.

### robot.model.stats module

**class** `robot.model.stats.Stat(name)`
**Bases:** `robot.utils.sortable.Sortable`

Generic statistic object used for storing all the statistic values.

**name = None**
Human readable identifier of the object these statistics belong to. Either All Tests or Critical Tests for *TotalStatistics*, long name of the suite for *SuiteStatistics* or name of the tag for *TagStatistics*

**passed = None**
Number of passed tests.

**failed = None**
Number of failed tests.

**elapsed = None**
Number of milliseconds it took to execute.

**get_attributes**(include_label=False, include_elapsed=False, exclude_empty=True, values_as_strings=False, html_escape=False)

**total**

**add_test**(test)

**visit**(visitor)
**class** robot.model.stats.TotalStat(name)
Bases: robot.model.stats.Stat
Stores statistic values for a test run.

type = 'total'

**add_test**(test)

**get_attributes**(include_label=False, include_elapsed=False, exclude_empty=True, values_as_strings=False, html_escape=False)

total

**visit**(visitor)

**class** robot.model.stats.SuiteStat(suite)
Bases: robot.model.stats.Stat
Stores statistics values for a single suite.

type = 'suite'

id = None
Identifier of the suite, e.g. s1-s2.

elsapsed = 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)
```python
class robot.model.suitestatistics.SuiteStatisticsBuilder(suite_stat_level)
    Bases: object
    current
    start_suite(suite)
    add_test(test)
    end_suite()

robot.model.tags module

class robot.model.tags.Tags(tags=None)
    Bases: object
    add(tags)
    remove(tags)
    match(tags)

class robot.model.tags.TagPatterns(patterns)
    Bases: object
    match(tags)

robot.model.tags.TagPattern(pattern)

class robot.model.tags.SingleTagPattern(pattern)
    Bases: object
    match(tags)

class robot.model.tags.AndTagPattern(patterns)
    Bases: object
    match(tags)

class robot.model.tags.OrTagPattern(patterns)
    Bases: object
    match(tags)

class robot.model.tags.NotTagPattern(must_match, *must_not_match)
    Bases: object
    match(tags)

robot.model.tagsetter module

class robot.model.tagsetter.TagSetter(add=None, remove=None)
    Bases: robot.model.visitor.SuiteVisitor
    start_suite(suite)
        Called when suite starts. Default implementation does nothing.
        Can return explicit False to stop visiting.
    visit_test(test)
        Implements traversing through the test and its keywords.
```

4.1. robot package
Can be overridden to allow modifying the passed in test without calling start_test() or end_test() nor visiting keywords.

visit_keyword(keyword)
Implements traversing through the keyword and its child keywords.
Can be overridden to allow modifying the passed in kw without calling start_keyword() or end_keyword() nor visiting child keywords.

end_keyword(keyword)
Called when keyword ends. Default implementation does nothing.

end_message(msg)
Called when message ends. Default implementation does nothing.

end_suite(suite)
Called when suite ends. Default implementation does nothing.

end_test(test)
Called when test ends. Default implementation does nothing.

start_keyword(keyword)
Called when keyword starts. Default implementation does nothing.
Can return explicit False to stop visiting.

start_message(msg)
Called when message starts. Default implementation does nothing.
Can return explicit False to stop visiting.

start_test(test)
Called when test starts. Default implementation does nothing.
Can return explicit False to stop visiting.

visit_message(msg)
Implements visiting the message.
Can be overridden to allow modifying the passed in msg without calling start_message() or end_message().

visit_suite(suite)
Implements traversing through the suite and its direct children.
Can be overridden to allow modifying the passed in suite without calling start_suite() or end_suite() nor visiting child suites, tests or keywords (setup and teardown) at all.

robot.model.tagstatistics module

class robot.model.tagstatistics.TagStatistics(critical_stats, non_critical_stats, combined_stats)

Bases: object

Container for tag statistics.

tags = None
Dictionary, where key is the name of the tag as a string and value is an instance of TagStat.

critical = None
List of CriticalTagStat objects.
non_critical = None
List of CriticalTagStat objects.

combined = None
List of CombinedTagStat objects.

visit(visitor)

class robot.model.tagstatistics.TagStatisticsBuilder
    (criticality=None, included=None, excluded=None, combined=None, docs=None, links=None)

Bases: object

add_test(test)

class robot.model.tagstatistics.TagStatInfo
    (docs=None, links=None)

Bases: object

get_stat(tag)

get_critical_stats(criticality, critical=True)

get_combined_stats(combined=None)

get_doc(tag)

get_links(tag)

class robot.model.tagstatistics.TagStatDoc
    (pattern, doc)

Bases: object

match(tag)

class robot.model.tagstatistics.TagStatLink
    (pattern, link, title)

Bases: object

match(tag)

get_link(tag)

robot.model.testcase module

class robot.model.testcase.TestCase
    (name=", doc=", tags=None, timeout=None)

Bases: robot.model.modelobject.ModelObject

Base model for a single test case.

Extended by robot.running.model.TestCase and robot.result.model.TestCase.

keyword_class
    alias of robot.model.keyword.Keyword

parent
    Parent suite.

name
    Test case name.

doc
    Test case documentation.

timeout
    Test case timeout.

4.1. robot package
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.

source

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.

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.

class robot.model.testcase.TestCases (test_class=<class 'robot.model.testcase.TestCase'>, parent=None, tests=None)
Bases: robot.model.itemlist.ItemList

append (item)
clear ()
count (item)
create (*args, **kwargs)
extend (items)
index (item, *start_and_end)
insert (index, item)
pop (*index)
remove (item)
reverse ()
sort ()
visit (visitor)

robot.model.testsuite module

class robot.model.testsuite.TestSuite (name="", doc="", metadata=None, source=None, rpa=False)

    Bases: robot.model.modelobject.ModelObject

    Base model for single suite.

    Extended by robot.running.model.TestSuite and robot.result.model.TestSuite.

    test_class
        alias of robot.model.testcase.TestCase

    keyword_class
        alias of robot.model.keyword.Keyword

    parent
        Parent suite. None with the root suite.

    doc
        Test suite documentation.

    source
        Path to the source file or directory.

    rpa

    name
        Test suite name. If not set, constructed from child suite names.

    longname
        Suite name prefixed with the long name of the parent suite.

    metadata
        Free test suite metadata as a dictionary.

    suites
        Child suites as a TestSuites object.

    tests
        Tests as a TestCases object.

    keywords
        Suite setup and teardown as a Keywords object.

    id
        An automatically generated unique id.

        The root suite has id s1, its child suites have ids s1-s1, s1-s2, . . . , their child suites get ids s1-s1-s1, s1-s1-s2, . . . , s1-s2-s1, . . . , and so on.

        The first test in a suite has an id like s1-t1, the second has an id s1-t2, and so on. Similarly keywords in suites (setup/teardown) and in tests get ids like s1-k1, s1-t1-k1, and s1-s4-t2-k5.

    test_count
        Number of the tests in this suite, recursively.

    has_tests

    set_tags (add=None, remove=None, persist=False)
        Add and/or remove specified tags to the tests in this suite.
Parameters

- **add** – Tags to add as a list or, if adding only one, as a single string.
- **remove** – Tags to remove as a list or as a single string. Can be given as patterns where * and ? work as wildcards.
- **persist** – Add/remove specified tags also to new tests added to this suite in the future.

**filter***(included_suites=None, included_tests=None, included_tags=None, excluded_tags=None)*

Select test cases and remove others from this suite.

Parameters have the same semantics as --suite, --test, --include, and --exclude command line options. All of them can be given as a list of strings, or when selecting only one, as a single string.

Child suites that contain no tests after filtering are automatically removed.

Example:
```
suite.filter(included_tests=['Test 1', '* Example'],
            included_tags='priority-1')
```

**configure** *(**options)**

A shortcut to configure a suite using one method call.

Can only be used with the root test suite.

- **Parameters options** – Passed to SuiteConfigurer that will then set suite attributes, call filter(), etc. as needed.

**copy** *(**attributes)**

Return shallow copy of this object.

- **Parameters attributes** – Attributes to be set for the returned copy automatically. For example, test.copy (name='New name').

See also deepcopy(). The difference between these two is the same as with the standard copy.copy and copy.deepcopy functions that these methods also use internally.

New in Robot Framework 3.0.1.

**deepcopy** *(**attributes)**

Return deep copy of this object.

- **Parameters attributes** – Attributes to be set for the returned copy automatically. For example, test.deepcopy (name='New name').

See also copy(). The difference between these two is the same as with the standard copy.copy and copy.deepcopy functions that these methods also use internally.

New in Robot Framework 3.0.1.

**remove_empty_suites** *(preserve_direct_children=False)*

Removes all child suites not containing any tests, recursively.

**visit** *(visitor)*

Visitor interface entry-point.

**class robot.model.testsuite.TestSuites**(suite_class=<class 'robot.model.testsuite.TestSuite'>, parent=None, suites=None)

Bases: robot.model.itemlist.ItemList

**append** *(item)*

**clear** ()
count(item)
create(*args, **kwargs)
extend(items)
index(item, *start_and_end)
insert(index, item)
pop(*index)
remove(item)
reverse()
sort()
visit(visitor)

robot.model.totalstatistics module

class robot.model.totalstatistics.TotalStatistics(rpa=False)
    Bases: object
    Container for total statistics.
    all = None
        Instance of TotalStat for all the tests.
    visit(visitor)
    message
        String representation of the statistics.
        For example:
        2 critical tests, 1 passed, 1 failed
        2 tests total, 1 passed, 1 failed

class robot.model.totalstatistics.TotalStatisticsBuilder(suite=None, rpa=False)
    Bases: robot.model.visitor.SuiteVisitor
    add_test(test)
    visit_test(test)
        Implements traversing through the test and its keywords.
        Can be overridden to allow modifying the passed in test without calling start_test() or end_test() nor visiting keywords.
    visit_keyword(kw)
        Implements traversing through the keyword and its child keywords.
        Can be overridden to allow modifying the passed in kw without calling start_keyword() or end_keyword() nor visiting child keywords.
    end_keyword(keyword)
        Called when keyword ends. Default implementation does nothing.
    end_message(msg)
        Called when message ends. Default implementation does nothing.

4.1. robot package
**Robot Framework Documentation, Release 3.2b3.dev1**

*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.

```python
"""Pre-run modifier that selects only every Xth test for execution.
Starts from the first test by default. Tests are selected per suite.
"""

from robot.api import SuiteVisitor
class SelectEveryXthTest(SuiteVisitor):

    def __init__(self, x, start=0):
        self.x = int(x)
        self.start = int(start)

    def start_suite(self, suite):
        """Modify suite's tests to contain only every Xth."""
        suite.tests = suite.tests[self.start::self.x]

    def end_suite(self, suite):
        """Remove suites that are empty after removing tests."""
        suite.suites = [s for s in suite.suites if s.test_count > 0]

    def visit_test(self, test):
        """Avoid visiting tests and their keywords to save a little time."""
        pass

For more examples it is possible to look at the source code of visitors used internally by Robot Framework itself. Some good examples are `TagSetter` and `keyword removers`.

**class robot.model.visitor.SuiteVisitor**

Bases: object

Abstract class to ease traversing through the test suite structure.

See the module level documentation for more information and an example.

**visit_suite(suite)**

Implements traversing through the suite and its direct children.

Can be overridden to allow modifying the passed in `suite` without calling `start_suite()` or `end_suite()` nor visiting child suites, tests or keywords (setup and teardown) at all.

**start_suite(suite)**

Called when suite starts. Default implementation does nothing.

4.1. robot package 193
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

```python
class robot.output.console.dotted.DottedOutput(width=78, colors='AUTO', std-out=None, stderr=None)
    Bases: object
    start_suite(suite)
end_test(test)
end_suite(suite)
message(msg)
output_file(name, path)
```

```python
class robot.output.console.dotted.StatusReporter(stream, width)
    Bases: robot.model.visitor.SuiteVisitor
    report(suite)
    visit_test(test)
        Implements traversing through the test and its keywords.
        Can be overridden to allow modifying the passed in test without calling start_test() or end_test() nor visiting keywords.
    end_keyword(keyword)
        Called when keyword ends. Default implementation does nothing.
    end_message(msg)
        Called when message ends. Default implementation does nothing.
    end_suite(suite)
        Called when suite ends. Default implementation does nothing.
    end_test(test)
        Called when test ends. Default implementation does nothing.
    start_keyword(keyword)
        Called when keyword starts. Default implementation does nothing.
        Can return explicit False to stop visiting.
    start_message(msg)
        Called when message starts. Default implementation does nothing.
        Can return explicit False to stop visiting.
    start_suite(suite)
        Called when suite starts. Default implementation does nothing.
        Can return explicit False to stop visiting.
    start_test(test)
        Called when test starts. Default implementation does nothing.
        Can return explicit False to stop visiting.
    visit_keyword(kw)
        Implements traversing through the keyword and its child keywords.
```

4.1. robot package
Can be overridden to allow modifying the passed in kw without calling \texttt{start\_keyword()} or \texttt{end\_keyword()} nor visiting child keywords.

\texttt{visit\_message}(\texttt{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}(\texttt{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.

\texttt{robot.output.console.highlighting} module

\texttt{class} \texttt{robot.output.console.highlighting.HighlightingStream(\texttt{stream}, \texttt{colors='AUTO'})}  
\textbf{Bases:} \texttt{object}  
\texttt{write}(\texttt{text}, \texttt{flush=True})  
\texttt{flush}()  
\texttt{highlight}(\texttt{text}, \texttt{status=None}, \texttt{flush=True})  
\texttt{error}(\texttt{message}, \texttt{level})

\texttt{robot.output.console.highlighting.Highlighter(\texttt{stream})}  
\texttt{class} \texttt{robot.output.console.highlighting.AnsiHighlighter(\texttt{stream})}  
\textbf{Bases:} \texttt{object}  
\texttt{green}()  
\texttt{red}()  
\texttt{yellow}()  
\texttt{reset}()  

\texttt{class} \texttt{robot.output.console.highlighting.NoHighlighting(\texttt{stream})}  
\textbf{Bases:} \texttt{robot.output.console.highlighting.AnsiHighlighter}  
\texttt{green}()  
\texttt{red}()  
\texttt{reset}()  
\texttt{yellow}()

\texttt{class} \texttt{robot.output.console.highlighting.DosHighlighter(\texttt{stream})}  
\textbf{Bases:} \texttt{object}  
\texttt{green}()  
\texttt{red}()  
\texttt{yellow}()  
\texttt{reset}()
robot.output.console.quiet module

class robot.output.console.quiet.QuietOutput(colors='AUTO', stderr=None)
    Bases: object
    message(msg)

class robot.output.console.quiet.NoOutput
    Bases: object

robot.output.console.verbose module

class robot.output.console.verbose.VerboseOutput(width=78, colors='AUTO',
    markers='AUTO', stdout=None, stderr=None)
    Bases: object
    start_suite(suite)
    end_suite(suite)
    start_test(test)
    end_test(test)
    start_keyword(kw)
    end_keyword(kw)
    message(msg)
    output_file(name, path)

class robot.output.console.verbose.VerboseWriter(width=78, colors='AUTO',
    markers='AUTO', stdout=None, stderr=None)
    Bases: object
    info(name, doc, start_suite=False)
    suite_separator()
    test_separator()
    status(status, clear=False)
    message(message)
    keyword_marker(status)
    error(message, level, clear=False)
    output(name, path)

class robot.output.console.verbose.KeywordMarker(highlighter, markers)
    Bases: object
    mark(status)
    reset_count()
Submodules

robot.output.debugfile module

robot.output.debugfile.DebugFile(path)

robot.output.filelogger module

class robot.output.filelogger.FileLogger(path, level)
    Bases: robot.output.loggerhelper.AbstractLogger
    message(msg)
    start_suite(suite)
    end_suite(suite)
    start_test(test)
    end_test(test)
    start_keyword(kw)
    end_keyword(kw)
    output_file(name, path)
    close()
    debug(msg)
    error(msg)
    fail(msg)
    info(msg)
    set_level(level)
    trace(msg)
    warn(msg)
    write(message, level, html=False)

robot.output.librarylogger module

Implementation of the public test library logging API.
This is exposed via robot.api.logger. Implementation must reside here to avoid cyclic imports.
robot.output.librarylogger.write(msg, level, html=False)
robot.output.librarylogger.trace(msg, html=False)
robot.output.librarylogger.debug(msg, html=False)
robot.output.librarylogger.info(msg, html=False, also_console=False)
robot.output.librarylogger.warn(msg, html=False)
robot.output.librarylogger.error(msg, html=False)
robot.output.librarylogger.console(msg, newline=True, stream='stdout')
robot.output.listenerarguments module

class robot.output.listenerarguments.ListenerArguments (arguments)
    Bases: object
        get_arguments (version)
        classmethod by_method_name (name, arguments)

class robot.output.listenerarguments.MessageArguments (arguments)
    Bases: robot.output.listenerarguments.ListenerArguments
        classmethod by_method_name (name, arguments)
        get_arguments (version)

class robot.output.listenerarguments.StartSuiteArguments (arguments)
    Bases: robot.output.listenerarguments._ListenerArgumentsFromItem
        classmethod by_method_name (name, arguments)
        get_arguments (version)

class robot.output.listenerarguments.EndSuiteArguments (arguments)
    Bases: robot.output.listenerarguments.StartSuiteArguments
        classmethod by_method_name (name, arguments)
        get_arguments (version)

class robot.output.listenerarguments.StartTestArguments (arguments)
    Bases: robot.output.listenerarguments._ListenerArgumentsFromItem
        classmethod by_method_name (name, arguments)
        get_arguments (version)

class robot.output.listenerarguments.EndTestArguments (arguments)
    Bases: robot.output.listenerarguments.StartTestArguments
        classmethod by_method_name (name, arguments)
        get_arguments (version)

class robot.output.listenerarguments.StartKeywordArguments (arguments)
    Bases: robot.output.listenerarguments._ListenerArgumentsFromItem
        classmethod by_method_name (name, arguments)
        get_arguments (version)

class robot.output.listenerarguments.EndKeywordArguments (arguments)
    Bases: robot.output.listenerarguments.StartKeywordArguments
        classmethod by_method_name (name, arguments)
        get_arguments (version)

robot.output.listenermethods module

class robot.output.listenermethods.ListenerMethods (method_name, listeners)
    Bases: object

class robot.output.listenermethods.LibraryListenerMethods (method_name)
    Bases: object
new_suite_scope()
discard_suite_scope()
register(listeners, library)
unregister(library)
class robot.output.listeners.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()
    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, …

cache_only

delayed_logging

log_message(msg)

   Messages about what the framework is doing, warnings, errors, …

log_output(output)

enable_library_import_logging()

disable_library_import_logging()

start_suite(suite)

end_suite(suite)

start_test(test)

end_test(test)

start_keyword(keyword)

end_keyword(keyword)

imported(import_type, name, **attrs)

output_file(file_type, path)

   Finished output, report, log, debug, or xunit file

close()

debug(msg)

error(msg)

fail(msg)

info(msg)

set_level(level)

trace(msg)

warn(msg)

4.1. robot package
write (message, level, html=False)

class robot.output.logger.LoggerProxy (logger, method_names=None, prefix=None)
    Bases: robot.output.loggerhelper.AbstractLoggerProxy

robot.output.loggerhelper module

class robot.output.loggerhelper.AbstractLogger (level='TRACE')
    Bases: object
        set_level (level)
        trace (msg)
        debug (msg)
        info (msg)
        warn (msg)
        fail (msg)
        error (msg)
        write (message, level, html=False)
        message (msg)

class robot.output.loggerhelper.Message (message, level='INFO', html=False, timestamp=None)
    Bases: robot.model.message.Message
        message
        resolve_delayed_message ()
        copy (**attributes)
            Return shallow copy of this object.

            Parameters attributes – Attributes to be set for the returned copy automatically. For example, test.copy (name='New name').

            See also deepcopy (). The difference between these two is the same as with the standard copy.copy and copy.deepcopy functions that these methods also use internally.

            New in Robot Framework 3.0.1.

        deepcopy (**attributes)
            Return deep copy of this object.

            Parameters attributes – Attributes to be set for the returned copy automatically. For example, test.deepcopy (name='New name').

            See also copy (). The difference between these two is the same as with the standard copy.copy and copy.deepcopy functions that these methods also use internally.

            New in Robot Framework 3.0.1.

        html
        html_message
            Returns the message content as HTML.

        level
        parent
timestamp

visit(visitor)

Visitor interface entry-point.

class robot.output.loggerhelper.IsLogged(level)
    Bases: object
    set_level(level)

class robot.output.loggerhelper.AbstractLoggerProxy(logger, method_names=None, prefix=None)
    Bases: object

robot.output.output module

class robot.output.output.Output(settings)
    Bases: robot.output.loggerhelper.AbstractLogger
    register_error_listener(listener)
    close(result)
    start_suite(suite)
    end_suite(suite)
    start_test(test)
    end_test(test)
    start_keyword(kw)
    end_keyword(kw)
    message(msg)
    set_log_level(level)
    debug(msg)
    error(msg)
    fail(msg)
    info(msg)
    set_level(level)
    trace(msg)
    warn(msg)
    write(message, level, html=False)

robot.output.pyloggingconf module

robot.output.pyloggingconf.robot_handler_enabled(*args, **kwds)
robot.output.pyloggingconf.set_level(level)

class robot.output.pyloggingconf.RobotHandler(level=0)
    Bases: logging.Handler
    Initializes the instance - basically setting the formatter to None and the filter list to empty.

4.1. robot package
emit (record)
Do whatever it takes to actually log the specified logging record.

This version is intended to be implemented by subclasses and so raises a NotImplementedError.

acquire()
Acquire the I/O thread lock.

addFilter (filter)
Add the specified filter to this handler.

close()
Tidy up any resources used by the handler.

This version removes the handler from an internal map of handlers, _handlers, which is used for handler lookup by name. Subclasses should ensure that this gets called from overridden close() methods.

createLock()
Acquire a thread lock for serializing access to the underlying I/O.

filter (record)
Determine if a record is loggable by consulting all the filters.

The default is to allow the record to be logged; any filter can veto this and the record is then dropped. Returns a zero value if a record is to be dropped, else non-zero.

flush()
Ensure all logging output has been flushed.

This version does nothing and is intended to be implemented by subclasses.

format (record)
Format the specified record.

If a formatter is set, use it. Otherwise, use the default formatter for the module.

get_name()

handle (record)
Conditionally emit the specified logging record.

Emission depends on filters which may have been added to the handler. Wrap the actual emission of the record with acquisition/release of the I/O thread lock. Returns whether the filter passed the record for emission.

handleError (record)
Handle errors which occur during an emit() call.

This method should be called from handlers when an exception is encountered during an emit() call. If raiseExceptions is false, exceptions get silently ignored. This is what is mostly wanted for a logging system - most users will not care about errors in the logging system, they are more interested in application errors. You could, however, replace this with a custom handler if you wish. The record which was being processed is passed in to this method.

name

release()
Release the I/O thread lock.

removeFilter (filter)
Remove the specified filter from this handler.

setFormatter (fmt)
Set the formatter for this handler.
setLevel (level)
   Set the logging level of this handler.

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', rpa=False, generator='Robot')
   Bases: robot.result.visitor.ResultVisitor
      close ()
      set_log_level (level)
      message (msg)
      log_message (msg)
      start_keyword (kw)
         Called when keyword starts. Default implementation does nothing.
         Can return explicit False to stop visiting.
      end_keyword (kw)
         Called when keyword ends. Default implementation does nothing.
      start_test (test)
         Called when test starts. Default implementation does nothing.
         Can return explicit False to stop visiting.
      end_test (test)
         Called when test ends. Default implementation does nothing.
      start_suite (suite)
         Called when suite starts. Default implementation does nothing.
         Can return explicit False to stop visiting.
      end_suite (suite)
         Called when suite ends. Default implementation does nothing.
      start_statistics (stats)
      end_statistics (stats)
      start_total_statistics (total_stats)
      end_total_statistics (total_stats)
      start_tag_statistics (tag_stats)
      end_tag_statistics (tag_stats)
      start_suite_statistics (tag_stats)
end_suite_statistics(tag_stats)

visit_stat(stat)

start_errors(errors=None)
end_errors(errors=None)

end_message(msg)
Called when message ends. Default implementation does nothing.

end_result(result)
end_stat(stat)

start_message(msg)
Called when message starts. Default implementation does nothing.

Can return explicit False to stop visiting.

start_result(result)
start_stat(stat)

visit_errors(errors)

visit_keyword(kw)
Implements traversing through the keyword and its child keywords.

Can be overridden to allow modifying the passed in kw without calling start_keyword() or end_keyword() nor visiting child keywords.

visit_message(msg)
Implements visiting the message.

Can be overridden to allow modifying the passed in msg without calling start_message() or end_message().

visit_result(result)

visit_statistics(stats)

visit_suite(suite)
Implements traversing through the suite and its direct children.

Can be overridden to allow modifying the passed in suite without calling start_suite() or end_suite() nor visiting child suites, tests or keywords (setup and teardown) at all.

visit_suite_statistics(stats)

visit_tag_statistics(stats)

visit_test(test)
Implements traversing through the test and its keywords.

Can be overridden to allow modifying the passed in test without calling start_test() or end_test() nor visiting keywords.

visit_total_statistics(stats)

robot.parsing package
Implements test data parsing.

Main entry points to parsing are the following:
• `get_tokens()` and `get_resource_tokens()` for parsing test data to tokens.
• `get_model()` and `get_resource_model()` for parsing test data into a model represented as an abstract syntax tree (AST).

TODO: Document how to modify the returned model using `ModelVisitor` and `ModelTransformer`. Also mention that the model as well as tokens are part of the public API.

Like with rest of the public API, these functions and classes are exposed also via the `robot.api` package.

The `robot.parsing` package has been totally rewritten in Robot Framework 3.2 and all code using it needs to be updated. Depending on the use case, it may be possible to use the higher level `TestSuiteBuilder()` that has not changed instead.

Example

```python
import ast
import sys

from robot.api import get_model

class TestNamePrinter(ast.NodeVisitor):
    def visit_File(self, node):
        print(f"File '{node.source}' has following tests:")
        # Must call 'generic_visit' to visit also child nodes.
        self.generic_visit(node)

    def visit_KeywordSection(self, node):
        # Overriding a visitor method without calling 'generic_visit'
        # prevents child nodes being visited. In this case it means
        # 'visit_Name' is not called with nodes in the keyword section.
        pass

    def visit_Name(self, node):
        print(f"- {node.name}")

model = get_model(sys.argv[1])
TestNamePrinter().visit(model)
```

TODO: Add example modifying model.

Subpackages

`robot.parsing.lexer` package

Submodules

`robot.parsing.lexer.context` module

```python
class robot.parsing.lexer.context.LexingContext (settings=None)
    Bases: object
```

4.1. robot package
settings_class = None

lex_setting (statement)

class robot.parsing.lexer.context.FileContext (settings=None)
    Bases: robot.parsing.lexer.context.LexingContext

    sections_class = None
    setting_section (statement)
    variable_section (statement)
    test_case_section (statement)
    keyword_section (statement)
    comment_section (statement)
    keyword_context ()
    lex_invalid_section (statement)
    lex_setting (statement)

    settings_class = None

class robot.parsing.lexer.context.TestCaseFileContext (settings=None)
    Bases: robot.parsing.lexer.context.FileContext

    sections_class
        alias of robot.parsing.lexer.sections.TestCaseFileSections

    settings_class
        alias of robot.parsing.lexer.settings.TestCaseFileSettings

    test_case_context ()
    comment_section (statement)
    keyword_context ()
    keyword_section (statement)
    lex_invalid_section (statement)
    lex_setting (statement)

    setting_section (statement)
    test_case_section (statement)

    variable_section (statement)

class robot.parsing.lexer.context.ResourceFileContext (settings=None)
    Bases: robot.parsing.lexer.context.FileContext

    sections_class
        alias of robot.parsing.lexer.sections.ResourceFileSections

    settings_class
        alias of robot.parsing.lexer.settings.ResourceFileSettings

    comment_section (statement)
    keyword_context ()
    keyword_section (statement)
lex_invalid_section (statement)
lex_setting (statement)
setting_section (statement)
test_case_section (statement)
variable_section (statement)

class robot.parsing.lexer.context.InitFileContext (settings=None)
  Bases: robot.parsing.lexer.context.FileContext
  sections_class
    alias of robot.parsing.lexer.sections.InitFileSections
  settings_class
    alias of robot.parsing.lexer.settings.InitFileSettings
  comment_section (statement)
  keyword_context ()
  keyword_section (statement)
  lex_invalid_section (statement)
  lex_setting (statement)
  setting_section (statement)
  test_case_section (statement)
  variable_section (statement)

class robot.parsing.lexer.context.TestCaseContext (settings=None)
  Bases: robot.parsing.lexer.context.LexingContext
  template_set
  lex_setting (statement)
  settings_class = None

class robot.parsing.lexer.context.KeywordContext (settings=None)
  Bases: robot.parsing.lexer.context.LexingContext
  template_set
  lex_setting (statement)
  settings_class = None

robot.parsing.lexer.lexers module

robot.parsing.lexer.readers module

robot.parsing.lexer.settings module

class robot.parsing.lexer.settings.Settings
  Bases: object
  names = ()
  aliases = ()
multi_use = ('Metadata', 'Library', 'Resource', 'Variables')
single_value = ('Resource', 'Test Timeout', 'Test Template', 'Timeout', 'Template')
name_and_arguments = ('Metadata', 'Suite Setup', 'Suite Teardown', 'Test Setup', 'Test Teardown', 'Test Template', 'Setup', 'Teardown', 'Template', 'Resource', 'Variables')
name_arguments_and_with_name = ('Library',)

class robot.parsing.lexer.settings.TestCaseFileSettings
    Bases: robot.parsing.lexer.settings.Settings
    names = ('Documentation', 'Metadata', 'Suite Setup', 'Suite Teardown', 'Test Setup', 'Test Teardown', 'Test Template', 'Test Timeout', 'Force Tags', 'Default Tags', 'Library', 'Resource', 'Variables')
    aliases = {'Task Setup': 'Test Setup', 'Task Teardown': 'Test Teardown', 'Task Template': 'Test Template', 'Task Timeout': 'Test Timeout'}
    lex(statement)
    multi_use = ('Metadata', 'Library', 'Resource', 'Variables')
    name_and_arguments = ('Metadata', 'Suite Setup', 'Suite Teardown', 'Test Setup', 'Test Teardown', 'Test Template', 'Setup', 'Teardown', 'Template', 'Resource', 'Variables')
    name_arguments_and_with_name = ('Library',)
    single_value = ('Resource', 'Test Timeout', 'Test Template', 'Timeout', 'Template')

class robot.parsing.lexer.settings.InitFileSettings
    Bases: robot.parsing.lexer.settings.Settings
    names = ('Documentation', 'Metadata', 'Suite Setup', 'Suite Teardown', 'Test Setup', 'Test Teardown', 'Test Timeout', 'Force Tags', 'Library', 'Resource', 'Variables')
    aliases = {}
    lex(statement)
    multi_use = ('Metadata', 'Library', 'Resource', 'Variables')
    name_and_arguments = ('Metadata', 'Suite Setup', 'Suite Teardown', 'Test Setup', 'Test Teardown', 'Test Template', 'Setup', 'Teardown', 'Template', 'Resource', 'Variables')
    name_arguments_and_with_name = ('Library',)
    single_value = ('Resource', 'Test Timeout', 'Test Template', 'Timeout', 'Template')

class robot.parsing.lexer.settings.ResourceFileSettings
    Bases: robot.parsing.lexer.settings.Settings
    names = ('Documentation', 'Library', 'Resource', 'Variables')
    aliases = {}
    lex(statement)
    multi_use = ('Metadata', 'Library', 'Resource', 'Variables')
    name_and_arguments = ('Metadata', 'Suite Setup', 'Suite Teardown', 'Test Setup', 'Test Teardown', 'Test Template', 'Setup', 'Teardown', 'Template', 'Resource', 'Variables')
    name_arguments_and_with_name = ('Library',)
    single_value = ('Resource', 'Test Timeout', 'Test Template', 'Timeout', 'Template')

class robot.parsing.lexer.settings.TestCaseSettings
    Bases: robot.parsing.lexer.settings.Settings
    names = ('Documentation', 'Tags', 'Setup', 'Teardown', 'Template', 'Timeout')
    template_set
    aliases = {}}
lex(statement)

multi_use = ('Metadata', 'Library', 'Resource', 'Variables')
name_and_arguments = ('Metadata', 'Suite Setup', 'Suite Teardown', 'Test Setup', 'Test name_arguments_and_with_name = ('Library',)
single_value = ('Resource', 'Test Timeout', 'Test Template', 'Timeout', 'Template')

class robot.parsing.lexer.settings.KeywordSettings
    Bases: robot.parsing.lexer.settings.Settings
    names = ('Documentation', 'Arguments', 'Teardown', 'Timeout', 'Tags', 'Return')
    aliases = {}

lex(statement)

multi_use = ('Metadata', 'Library', 'Resource', 'Variables')
name_and_arguments = ('Metadata', 'Suite Setup', 'Suite Teardown', 'Test Setup', 'Test name_arguments_and_with_name = ('Library',)
single_value = ('Resource', 'Test Timeout', 'Test Template', 'Timeout', 'Template')

robot.parsing.lexer.splitter module

robot.parsing.lexer.tokens module

class robot.parsing.lexer.tokens.Token(type, value=", lineno=-1, col_offset=-1, error=None)
    Bases: object

FIXME: Add documentation to Token class and types.

SETTING_HEADER = 'SETTING_HEADER'
VARIABLE_HEADER = 'VARIABLE_HEADER'
TESTCASE_HEADER = 'TESTCASE_HEADER'
KEYWORD_HEADER = 'KEYWORD_HEADER'
COMMENT_HEADER = 'COMMENT_HEADER'
TESTCASE_NAME = 'TESTCASE_NAME'
KEYWORD_NAME = 'KEYWORD_NAME'
DOCUMENTATION = 'DOCUMENTATION'
SUITE_SETUP = 'SUITE_SETUP'
SUITE_TEARDOWN = 'SUITE_TEARDOWN'
METADATA = 'METADATA'
TEST_SETUP = 'TEST_SETUP'
TEST_TEARDOWN = 'TEST_TEARDOWN'
TEST_TEMPLATE = 'TEST_TEMPLATE'
TEST_TIMEOUT = 'TEST_TIMEOUT'

4.1. robot package
FORCE_TAGS = 'FORCE_TAGS'
DEFAULT_TAGS = 'DEFAULT_TAGS'
LIBRARY = 'LIBRARY'
RESOURCE = 'RESOURCE'
VARIABLES = 'VARIABLES'
SETUP = 'SETUP'
TEARDOWN = 'TEARDOWN'
TEMPLATE = 'TEMPLATE'
TIMEOUT = 'TIMEOUT'
TAGS = 'TAGS'
ARGUMENTS = 'ARGUMENTS'
RETURN = 'RETURN'
NAME = 'NAME'
VARIABLE = 'VARIABLE'
ARGUMENT = 'ARGUMENT'
ASSIGN = 'ASSIGN'
KEYWORD = 'KEYWORD'
WITH_NAME = 'WITH_NAME'
FOR = 'FOR'
FOR_SEPARATOR = 'FOR_SEPARATOR'
OLD_FOR_INDENT = 'OLD_FOR_INDENT'
END = 'END'
SEPARATOR = 'SEPARATOR'
COMMENT = 'COMMENT'
CONTINUATION = 'CONTINUATION'
IGNORE = 'IGNORE'
EOL = 'EOL'
EOS = 'EOS'
ERROR = 'ERROR'
FATAL_ERROR = 'FATAL_ERROR'
DATA = 'DATA'
NON_DATA_TOKENS = ('SEPARATOR', 'COMMENT', 'CONTINUATION', 'IGNORE', 'EOL', 'EOS')
SETTING_TOKENS = ('DOCUMENTATION', 'SUITE_SETUP', 'SUITE_TEARDOWN', 'METADATA', 'TEST_SETUP', 'TEST_TEARDOWN', ...
HEADER_TOKENS = ('SETTING_HEADER', 'VARIABLE_HEADER', 'TESTCASE_HEADER', 'KEYWORD_HEADER')
type
value
lineno
col_offset
error
end_col_offset
set_error(error, fatal=False)
class robot.parsing.lexer.tokens.EOS(line=-1, col=-1)
Bases: robot.parsing.lexer.tokens.Token
classmethod from_token(token)
ARGUMENT = 'ARGUMENT'
ARGUMENTS = 'ARGUMENTS'
ASSIGN = 'ASSIGN'
COMMENT = 'COMMENT'
COMMENT_HEADER = 'COMMENT_HEADER'
CONTINUATION = 'CONTINUATION'
DATA = 'DATA'
DEFAULT_TAGS = 'DEFAULT_TAGS'
DOCUMENTATION = 'DOCUMENTATION'
END = 'END'
EOL = 'EOL'
EOS = 'EOS'
ERROR = 'ERROR'
FATAL_ERROR = 'FATAL_ERROR'
FOR = 'FOR'
FORCE_TAGS = 'FORCE_TAGS'
FOR_SEPARATOR = 'FOR_SEPARATOR'
HEADER_TOKENS = ('SETTING_HEADER', 'VARIABLE_HEADER', 'TESTCASE_HEADER', 'KEYWORD_HEADER')
IGNORE = 'IGNORE'
KEYWORD = 'KEYWORD'
KEYWORD_HEADER = 'KEYWORD_HEADER'
KEYWORD_NAME = 'KEYWORD_NAME'
LIBRARY = 'LIBRARY'
METADATA = 'METADATA'
NAME = 'NAME'
NON_DATA_TOKENS = ('SEPARATOR', 'COMMENT', 'CONTINUATION', 'IGNORE', 'EOL', 'EOS')
OLD_FOR_INDENT = 'OLD_FOR_INDENT'
RESOURCE = 'RESOURCE'

4.1. robot package
RETURN = 'RETURN'
SEPARATOR = 'SEPARATOR'
SETTING_HEADER = 'SETTING_HEADER'
SETTING_TOKENS = ('DOCUMENTATION', 'SUITE_SETUP', 'SUITE_TEARDOWN', 'METADATA', 'TEST_SETUP', 'TEST_TEARDOWN', 'TEMPLATE', 'TIMEOUT', 'TAGS', 'ARGUMENTS', 'RETURN')
SETUP = 'SETUP'
SUITE_SETUP = 'SUITE_SETUP'
SUITE_TEARDOWN = 'SUITE_TEARDOWN'
TAGS = 'TAGS'
TEARDOWN = 'TEARDOWN'
TEMPLATE = 'TEMPLATE'
TESTCASE_HEADER = 'TESTCASE_HEADER'
TESTCASE_NAME = 'TESTCASE_NAME'
TEST_SETUP = 'TEST_SETUP'
TEST_TEARDOWN = 'TEST_TEARDOWN'
TEST_TEMPLATE = 'TEST_TEMPLATE'
TEST_TIMEOUT = 'TEST_TIMEOUT'
TIMEOUT = 'TIMEOUT'
VARIABLE = 'VARIABLE'
VARIABLES = 'VARIABLES'
VARIABLE_HEADER = 'VARIABLE_HEADER'
WITH_NAME = 'WITH_NAME'
col_offset
end_col_offset
error
lineno
set_error (error, fatal=False)
type
value

robot.parsing.model package

Submodules

robot.parsing.model.blocks module

class robot.parsing.model.blocks.Block
    Bases: _ast.AST
    lineno
class robot.parsing.model.blocks.File(sections=None, source=None)
    Bases: robot.parsing.model.blocks.Block
    save(output=None)
    col_offset
    end_col_offset
    end_lineno
    lineno

class robot.parsing.model.blocks.Section(header=None, body=None)
    Bases: robot.parsing.model.blocks.Block
    col_offset
    end_col_offset
    end_lineno
    lineno

class robot.parsing.model.blocks.SettingSection(header=None, body=None)
    Bases: robot.parsing.model.blocks.Section
    col_offset
    end_col_offset
    end_lineno
    lineno

class robot.parsing.model.blocks.VariableSection(header=None, body=None)
    Bases: robot.parsing.model.blocks.Section
    col_offset
    end_col_offset
    end_lineno
    lineno

class robot.parsing.model.blocks.TestCaseSection(header=None, body=None)
    Bases: robot.parsing.model.blocks.Section
tasks
    col_offset
    end_col_offset
    end_lineno
    lineno

class robot.parsing.model.blocks.KeywordSection(header=None, body=None)
    Bases: robot.parsing.model.blocks.Section
    col_offset
```python
class robot.parsing.model.blocks.CommentSection(header=None, body=None):
    Bases: robot.parsing.model.blocks.Section

    col_offset
    end_col_offset
    end_lineno
    lineno

class robot.parsing.model.blocks.Body(items=None):
    Bases: robot.parsing.model.blocks.Block

    add(item)
    col_offset
    end_col_offset
    end_lineno
    lineno

class robot.parsing.model.blocks.TestCase(header, body=None):
    Bases: robot.parsing.model.blocks.Block

    name
    col_offset
    end_col_offset
    end_lineno
    lineno

class robot.parsing.model.blocks.Keyword(header, body=None):
    Bases: robot.parsing.model.blocks.Block

    name
    col_offset
    end_col_offset
    end_lineno
    lineno

class robot.parsing.model.blocks.ForLoop(header, body=None, end=None):
    Bases: robot.parsing.model.blocks.Block

    variables
    values
    flavor
    col_offset
    end_col_offset
    end_lineno
```
class robot.parsing.model.blocks.ModelWriter(output)
    Bases: robot.parsing.model.visitor.ModelVisitor
    write(model)
    visit_Statement(statement)
    generic_visit(node)
        Called if no explicit visitor function exists for a node.
    visit(node)
        Visit a node.

class robot.parsing.model.blocks.FirstStatementFinder
    Bases: robot.parsing.model.visitor.ModelVisitor
    classmethod find_from(model)
    visit_Statement(statement)
    generic_visit(node)
        Called if no explicit visitor function exists for a node.
    visit(node)
        Visit a node.

class robot.parsing.model.blocks.LastStatementFinder
    Bases: robot.parsing.model.visitor.ModelVisitor
    classmethod find_from(model)
    generic_visit(node)
        Called if no explicit visitor function exists for a node.
    visit(node)
        Visit a node.
    visit_Statement(statement)

robot.parsing.model.statements module

class robot.parsing.model.statements.Statement(tokens)
    Bases: _ast.AST
    type = None
    lineno
    col_offset
    end_lineno
    end_col_offset
    classmethod register(subcls)
    classmethod from_tokens(tokens)
    data_tokens
    get_value(type, default=None)
    get_token(type)
get_values(*types)
get_tokens(*types)
lines
error
class robot.parsing.model.statements.DocumentationOrMetadata(tokens)
    Bases: robot.parsing.model.statements.Statement
    col_offset
data_tokens
end_col_offset
end_lineno
error
classmethod from_tokens(tokens)
get_token(type)
get_tokens(*types)
get_value(type, default=None)
get_values(*types)
lineno
lines
classmethod register(subcls)
type = None
class robot.parsing.model.statements.SingleValue(tokens)
    Bases: robot.parsing.model.statements.Statement
    value
    col_offset
data_tokens
    end_col_offset
    end_lineno
    error
classmethod from_tokens(tokens)
    get_token(type)
    get_tokens(*types)
    get_value(type, default=None)
    get_values(*types)
    lineno
    lines
classmethod register(subcls)
type = None
```python
class robot.parsing.model.statements.MultiValue(tokens)
    Bases: robot.parsing.model.statements.Statement
    values
    col_offset
    data_tokens
    end_col_offset
    end_lineno
    error
    classmethod from_tokens(tokens)
    get_token(type)
    get_tokens(*types)
    get_value(type, default=None)
    get_values(*types)
    lineno
    lines
    classmethod register(subcls)
    type = None
class robot.parsing.model.statements.Fixture(tokens)
    Bases: robot.parsing.model.statements.Statement
    name
    args
    col_offset
    data_tokens
    end_col_offset
    end_lineno
    error
    classmethod from_tokens(tokens)
    get_token(type)
    get_tokens(*types)
    get_value(type, default=None)
    get_values(*types)
    lineno
    lines
    classmethod register(subcls)
    type = None
class robot.parsing.model.statements.SectionHeader(tokens)
    Bases: robot.parsing.model.statements.Statement
```

4.1. robot package 219
value
col_offset
data_tokens
der_offset
der_lineno
error
classmethod from_tokens(tokens)
get_token(type)
get_tokens(*types)
get_value(type, default=None)
get_values(*types)
lineno
lines
classmethod register(subcls)
type = None
class robot.parsing.model.statements.SettingSectionHeader(tokens)
    Bases: robot.parsing.model.statements.SectionHeader
type = 'SETTING_HEADER'
col_offset
data_tokens
der_offset
der_lineno
error
classmethod from_tokens(tokens)
get_token(type)
get_tokens(*types)
get_value(type, default=None)
get_values(*types)
lineno
lines
classmethod register(subcls)
value
class robot.parsing.model.statements.VariableSectionHeader(tokens)
    Bases: robot.parsing.model.statements.SectionHeader
type = 'VARIABLE_HEADER'
col_offset
data_tokens
end_col_offset
end_lineno
error
classmethod from_tokens(tokens)
get_token(type)
get_tokens(**types)
get_value(type, default=None)
get_values(**types)
lineno
lines
classmethod register(subcls)
value
class robot.parsing.model.statements.TestCaseSectionHeader(tokens)
Bases: robot.parsing.model.statements.SectionHeader
type = 'TESTCASE_HEADER'
col_offset
data_tokens
end_col_offset
end_lineno
error
classmethod from_tokens(tokens)
get_token(type)
get_tokens(**types)
get_value(type, default=None)
get_values(**types)
lineno
lines
classmethod register(subcls)
value
class robot.parsing.model.statements.KeywordSectionHeader(tokens)
Bases: robot.parsing.model.statements.SectionHeader
type = 'KEYWORD_HEADER'
col_offset
data_tokens
end_col_offset
end_lineno
error
classmethod from_tokens(tokens)
get_token(type)
get_tokens(*types)
get_value(type, default=None)
get_values(*types)
lineno
lines

classmethod register(subcls)

value
class robot.parsing.model.statements.CommentSectionHeader(tokens)
   Bases: robot.parsing.model.statements.SectionHeader
type = 'COMMENT_HEADER'
col_offset
data_tokens
end_col_offset
end_lineno
error

classmethod from_tokens(tokens)
get_token(type)
get_tokens(*types)
get_value(type, default=None)
get_values(*types)
lineno
lines

classmethod register(subcls)

value
class robot.parsing.model.statements.LibraryImport(tokens)
   Bases: robot.parsing.model.statements.Statement
type = 'LIBRARY'
name
args
alias
col_offset
data_tokens
end_col_offset
end_lineno
error
classmethod from_tokens(tokens)
get_token(type)
get_tokens(*types)
get_value(type, default=None)
get_values(*types)
lineno
lines

classmethod register(subcls)

class robot.parsing.model.statements.ResourceImport(tokens)
    Bases: robot.parsing.model.statements.Statement
    type = 'RESOURCE'
    name
    col_offset
    data_tokens
    end_col_offset
    end_lineno
    error
    classmethod from_tokens(tokens)
    get_token(type)
    get_tokens(*types)
    get_value(type, default=None)
    get_values(*types)
    lineno
    lines
    classmethod register(subcls)

class robot.parsing.model.statements.VariablesImport(tokens)
    Bases: robot.parsing.model.statements.Statement
    type = 'VARIABLES'
    name
    args
    col_offset
    data_tokens
    end_col_offset
    end_lineno
    error
    classmethod from_tokens(tokens)
    get_token(type)
get_tokens(*types)
get_value(type, default=None)
get_values(*types)
lineno
lines
classmethod register(subcls)

class robot.parsing.model.statements.Documentation(tokens)
Bases: robot.parsing.model.statements.DocumentationOrMetadata
type = 'DOCUMENTATION'
value
col_offset
data_tokens
end_col_offset
end_lineno
error
classmethod from_tokens(tokens)
get_token(type)
get_tokens(*types)
get_value(type, default=None)
get_values(*types)
lineno
lines
classmethod register(subcls)

class robot.parsing.model.statements.Metadata(tokens)
Bases: robot.parsing.model.statements.DocumentationOrMetadata
type = 'METADATA'
name
value
col_offset
data_tokens
end_col_offset
end_lineno
error
classmethod from_tokens(tokens)
get_token(type)
get_tokens(*types)
get_value(type, default=None)
```python
get_values(*types)
lineno
lines
classmethod register(subcls)
class robot.parsing.model.statements.ForceTags(tokens)
    Bases: robot.parsing.model.statements.MultiValue
type = 'FORCE_TAGS'
col_offset
data_tokens
end_col_offset
end_lineno
error
classmethod from_tokens(tokens)
get_token(type)
get_tokens(*types)
get_value(type, default=None)
get_values(*types)
lineno
lines
classmethod register(subcls)
values
class robot.parsing.model.statements.DefaultTags(tokens)
    Bases: robot.parsing.model.statements.MultiValue
type = 'DEFAULT_TAGS'
col_offset
data_tokens
end_col_offset
end_lineno
error
classmethod from_tokens(tokens)
get_token(type)
get_tokens(*types)
get_value(type, default=None)
get_values(*types)
lineno
lines
classmethod register(subcls)
```

4.1. robot package   225
values

class robot.parsing.model.statements.SuiteSetup(tokens)
    Bases: robot.parsing.model.statements.Fixture
    type = 'SUITE_SETUP'
    args
col_offset
data_tokens
end_col_offset
end_lineno
error
classmethod from_tokens(tokens)
get_token(type)
get_tokens(*types)
get_value(type, default=None)
get_values(*types)
lineno
lines
name
classmethod register(subcls)

class robot.parsing.model.statements.SuiteTeardown(tokens)
    Bases: robot.parsing.model.statements.Fixture
    type = 'SUITE_TEARDOWN'
    args
col_offset
data_tokens
end_col_offset
end_lineno
error
classmethod from_tokens(tokens)
get_token(type)
get_tokens(*types)
get_value(type, default=None)
get_values(*types)
lineno
lines
name
classmethod register(subcls)
class robot.parsing.model.statements.TestSetup(tokens)
    Bases: robot.parsing.model.statements.Fixture
    
    type = 'TEST_SETUP'
    args
    col_offset
    data_tokens
    end_col_offset
    end_lineno
    error
    classmethod from_tokens(tokens)
    get_token(type)
    get_tokens(*types)
    get_value(type, default=None)
    get_values(*types)
    lineno
    lines
    name
    classmethod register(subcls)

class robot.parsing.model.statements.TestTeardown(tokens)
    Bases: robot.parsing.model.statements.Fixture
    
    type = 'TEST_TEARDOWN'
    args
    col_offset
    data_tokens
    end_col_offset
    end_lineno
    error
    classmethod from_tokens(tokens)
    get_token(type)
    get_tokens(*types)
    get_value(type, default=None)
    get_values(*types)
    lineno
    lines
    name
    classmethod register(subcls)
class robot.parsing.model.statements.TestTemplate(tokens)
    Bases: robot.parsing.model.statements.SingleValue
    
    type = 'TEST_TEMPLATE'
    col_offset
data_tokens
    end_col_offset
    end_lineno
    error
    classmethod from_tokens(tokens)
    get_token(type)
    get_tokens(*types)
    get_value(type, default=None)
    get_values(*types)
    lineno
    lines
    classmethod register(subcls)
    value

class robot.parsing.model.statements.TestTimeout(tokens)
    Bases: robot.parsing.model.statements.SingleValue
    
    type = 'TEST_TIMEOUT'
    col_offset
data_tokens
    end_col_offset
    end_lineno
    error
    classmethod from_tokens(tokens)
    get_token(type)
    get_tokens(*types)
    get_value(type, default=None)
    get_values(*types)
    lineno
    lines
    classmethod register(subcls)
    value

class robot.parsing.model.statements.Variable(tokens)
    Bases: robot.parsing.model.statements.Statement
    
    type = 'VARIABLE'
name
value
col_offset
data_tokens
end_col_offset
end_lineno
error
classmethod from_tokens(tokens)
get_token(type)
get_tokens(*types)
get_value(type, default=None)
get_values(*types)
lineno
lines
classmethod register(subcls)
class robot.parsing.model.statements.TestCaseName(tokens)
Bases: robot.parsing.model.statements.Statement
type = 'TESTCASE_NAME'
name
col_offset
data_tokens
end_col_offset
end_lineno
error
classmethod from_tokens(tokens)
get_token(type)
get_tokens(*types)
get_value(type, default=None)
get_values(*types)
lineno
lines
classmethod register(subcls)
class robot.parsing.model.statements.KeywordName(tokens)
Bases: robot.parsing.model.statements.Statement
type = 'KEYWORD_NAME'
name
col_offset
data_tokens
data_tokens
end_col_offset
data_tokens
end_lineno
data_tokens
error
classmethod from_tokens(tokens)
classmethod from_tokens(tokens)
get_token(type)
get_token(type)
get_tokens(*types)
get_tokens(*types)
get_value(type, default=None)
get_value(type, default=None)
get_values(*types)
get_values(*types)
lineno
lineno
lines
lines
classmethod register(subcls)
classmethod register(subcls)
class robot.parsing.model.statements.Setup(tokens)
class robot.parsing.model.statements.Setup(tokens)
Bases: robot.parsing.model.statements.Fixture
Bases: robot.parsing.model.statements.Fixture
type = 'SETUP'
type = 'SETUP'
args
args
col_offset
col_offset
data_tokens
data_tokens
end_col_offset
end_col_offset
end_lineno
end_lineno
error
classmethod from_tokens(tokens)
classmethod from_tokens(tokens)
get_token(type)
get_token(type)
get_tokens(*types)
get_tokens(*types)
get_value(type, default=None)
get_value(type, default=None)
get_values(*types)
get_values(*types)
lineno
lineno
lines
lines
name
name
classmethod register(subcls)
classmethod register(subcls)
class robot.parsing.model.statements.Teardown(tokens)
class robot.parsing.model.statements.Teardown(tokens)
Bases: robot.parsing.model.statements.Fixture
Bases: robot.parsing.model.statements.Fixture
type = 'TEARDOWN'
type = 'TEARDOWN'
args
args
col_offset
col_offset
data_tokens
data_tokens
end_col_offset
end_col_offset
end_lineno

token
classmethod from_tokens(tokens)
get_token(type)
get_tokens(*types)
get_value(type, default=None)
get_values(*types)

lineno
lines
name
classmethod register(subcls)

class robot.parsing.model.statements.Tags(tokens)
Bases: robot.parsing.model.statements.MultiValue
type = 'TAGS'
col_offset
data_tokens
end_col_offset
end_lineno
error
classmethod from_tokens(tokens)
get_token(type)
get_tokens(*types)
get_value(type, default=None)
get_values(*types)

lineno
lines
classmethod register(subcls)
values
class robot.parsing.model.statements.Template(tokens)
Bases: robot.parsing.model.statements.SingleValue
type = 'TEMPLATE'
col_offset
data_tokens
end_col_offset
end_lineno
error
classmethod from_tokens(tokens)
get_token(type)
get_tokens(*types)
get_value(type, default=None)
get_values(*types)
lineno
lines
classmethod register(subcls)
value
class robot.parsing.model.statements.Timeout(tokens)
    Bases: robot.parsing.model.statements.SingleValue
    type = 'TIMEOUT'
    col_offset
data_tokens
end_col_offset
end_lineno
error
classmethod from_tokens(tokens)
get_token(type)
get_tokens(*types)
get_value(type, default=None)
get_values(*types)
lineno
lines
classmethod register(subcls)
value
class robot.parsing.model.statements.Arguments(tokens)
    Bases: robot.parsing.model.statements.MultiValue
    type = 'ARGUMENTS'
    col_offset
data_tokens
end_col_offset
end_lineno
error
classmethod from_tokens(tokens)
get_token(type)
get_tokens(*types)
get_value(type, default=None)
get_values(*types)
lineno
lines
classmethod register(subcls)
values
class robot.parsing.model.statements.Return(tokens)
Bases: robot.parsing.model.statements.MultiValue
type = 'RETURN'
col_offset
data_tokens
end_col_offset
end_lineno
error
classmethod from_tokens(tokens)
get_token(type)
get_tokens(*types)
get_value(type, default=None)
get_values(*types)
lineno
lines
classmethod register(subcls)
values
class robot.parsing.model.statements.KeywordCall(tokens)
Bases: robot.parsing.model.statements.Statement
type = 'KEYWORD'
keyword
args
assign
col_offset
data_tokens
end_col_offset
end_lineno
error
classmethod from_tokens(tokens)
get_token(type)
get_tokens(*types)
get_value(type, default=None)
get_values(*types)
lineno
lines
classmethod register(subcls)

class robot.parsing.model.statements.TemplateArguments(tokens)
    Bases: robot.parsing.model.statements.Statement
    type = 'ARGUMENT'
    args
col_offset
data_tokens
end_col_offset
end_lineno
error
classmethod from_tokens(tokens)

get_token(type)
get_tokens(*types)
get_value(type, default=None)
get_values(*types)
lineno
lines
classmethod register(subcls)

class robot.parsing.model.statements.ForLoopHeader(tokens)
    Bases: robot.parsing.model.statements.Statement
    type = 'FOR'
variables
values
flavor
col_offset
data_tokens
end_col_offset
end_lineno
error
classmethod from_tokens(tokens)

get_token(type)
get_tokens(*types)
get_value(type, default=None)
get_values(*types)
class robot.parsing.model.statements.End(tokens)
    Bases: robot.parsing.model.statements.Statement

    type = 'END'
    value
    col_offset
    data_tokens
    end_col_offset
    end_lineno
    error

    classmethod from_tokens(tokens)

    get_token(type)
    get_tokens(*types)
    get_value(type, default=None)
    get_values(*types)
    lineno
    lines

class robot.parsing.model.statements.Comment(tokens)
    Bases: robot.parsing.model.statements.Statement

    type = 'COMMENT'
    col_offset
    data_tokens
    end_col_offset
    end_lineno
    error

    classmethod from_tokens(tokens)

    get_token(type)
    get_tokens(*types)
    get_value(type, default=None)
    get_values(*types)
    lineno
    lines

classmethod register(subcls)
class robot.parsing.model.statements.Error(tokens)
    Bases: robot.parsing.model.statements.Statement
    type = 'ERROR'
    col_offset
data_tokens
    end_col_offset
    end_lineno
    error
classmethod from_tokens(tokens)
get_token(type)
get_tokens(*types)
get_value(type, default=None)
get_values(*types)
    lineno
    lines
classmethod register(subcls)

class robot.parsing.model.statements.EmptyLine(tokens)
    Bases: robot.parsing.model.statements.Statement
    type = 'EOL'
classmethod from_value(value)
    col_offset
data_tokens
    end_col_offset
    end_lineno
    error
classmethod from_tokens(tokens)
get_token(type)
get_tokens(*types)
get_value(type, default=None)
get_values(*types)
    lineno
    lines
classmethod register(subcls)
**robot.parsing.model.visitor module**

class robot.parsing.model.visitor.VisitorFinder
   Bases: object

class robot.parsing.model.visitor.ModelVisitor
      visit (node)
         Visit a node.
      generic_visit (node)
         Called if no explicit visitor function exists for a node.

class robot.parsing.model.visitor.ModelTransformer
      visit (node)
         Visit a node.
      generic_visit (node)
         Called if no explicit visitor function exists for a node.

**Submodules**

**robot.parsing.builders module**

**robot.parsing.suitestructure module**

class robot.parsing.suitestructure.SuiteStructure (source=None, init_file=None, children=None)
   Bases: object
   is_directory
   visit (visitor)

class robot.parsing.suitestructure.SuiteStructureBuilder (included_extensions=('robot', ), included_suites=None)
   Bases: object
   ignored_prefixes = ('_', '.')
   ignored_dirs = ('CVS', )
   build (paths)

class robot.parsing.suitestructure.SuiteStructureVisitor
   Bases: object
   visit_file (structure)
   visit_directory (structure)
   start_directory (structure)
   end_directory (structure)
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.expandkeywordmatcher module

class robot.reporting.expandkeywordmatcher.ExpandKeywordMatcher(expand_keywords)
    Bases: object
    
    match(kw)

robot.reporting.jsbuildingcontext module

class robot.reporting.jsbuildingcontext.JsBuildingContext(log_path=None, 
    split_log=False, expand_keywords=None, 
    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)
    check_expansion(kw)
    expand_keywords
    link(msg)
    strings
    start_splitting_if_needed(split=False)
    end_splitting(model)
    prune_input(**kwds)
robot.reporting.jsexecutionresult module

class robot.reporting.jsexecutionresult.JsExecutionResult (suite, statistics, errors, basemillis=None, split_results=None, min_level=None, expand_keywords=None)

    Bases: object
    remove_data_not_needed_in_report()

robot.reporting.jsmodelbuilders module

class robot.reporting.jsmodelbuilders.JsModelBuilder (log_path=None, split_log=False, expand_keywords=None, 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

  usage = 'log'

  write(path, config)

class robot.reporting.logreportwriters.ReportWriter(js_model)

  Bases: robot.reporting.logreportwriters._LogReportWriter

  usage = 'report'

  write(path, config)

class robot.reporting.logreportwriters.RobotModelWriter(output, model, config)

  Bases: robot.htmldata.htmlfilewriter.ModelWriter

  write(line)

  handles(line)

robot.reporting.outputwriter module

class robot.reporting.outputwriter.OutputWriter(output, rpa=False)

  Bases: robot.output.xmllogger.XmlLogger

  start_message(msg)

      Called when message starts. Default implementation does nothing.

      Can return explicit False to stop visiting.

  visit_keyword(kw)

      Implements traversing through the keyword and its child keywords.

      Can be overridden to allow modifying the passed in kw without calling start_keyword() or end_keyword() nor visiting child keywords.

  close()
end_result (result)
end_errors (errors=None)
end_keyword (kw)
    Called when keyword ends. Default implementation does nothing.
end_message (msg)
    Called when message ends. Default implementation does nothing.
end_stat (stat)
end_statistics (stats)
end_suite (suite)
    Called when suite ends. Default implementation does nothing.
end_suite_statistics (tag_stats)
end_tag_statistics (tag_stats)
end_test (test)
    Called when test ends. Default implementation does nothing.
end_total_statistics (total_stats)
log_message (msg)
message (msg)
set_log_level (level)
start_errors (errors=None)
start_keyword (kw)
    Called when keyword starts. Default implementation does nothing.
        Can return explicit False to stop visiting.
start_result (result)
start_stat (stat)
start_statistics (stats)
start_suite (suite)
    Called when suite starts. Default implementation does nothing.
        Can return explicit False to stop visiting.
start_suite_statistics (tag_stats)
start_tag_statistics (tag_stats)
start_test (test)
    Called when test starts. Default implementation does nothing.
        Can return explicit False to stop visiting.
start_total_statistics (total_stats)
visit_errors (errors)
visit_message (msg)
    Implements visiting the message.
        Can be overridden to allow modifying the passed in msg without calling start_message() or end_message().
visit_result (result)
visit_stat (stat)
visit_statistics (stats)
visit_suite (suite)
  Implements traversing through the suite and its direct children.
  Can be overridden to allow modifying the passed in suite without calling start_suite() or end_suite() nor visiting child suites, tests or keywords (setup and teardown) at all.
visit_suite_statistics (stats)
visit_tag_statistics (stats)
visit_test (test)
  Implements traversing through the test and its keywords.
  Can be overridden to allow modifying the passed in test without calling start_test() or end_test() nor visiting keywords.
visit_total_statistics (stats)

robot.reporting.resultwriter module

class robot.reporting.resultwriter.ResultWriter (*sources)
  Bases: object
  A class to create log, report, output XML and xUnit files.
  
  Parameters sources – Either one Result object, or one or more paths to existing output XML files.

  By default writes report.html and log.html, but no output XML or xUnit files. Custom file names can be given and results disabled or enabled using settings or options passed to the write_results() method. The latter is typically more convenient:

  ```python
  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)
  
robot.reporting.stringcache module

class robot.reporting.stringcache.StringIndex
  Bases: int
**bit_length()** → int
   Number of bits necessary to represent self in binary. >>> bin(37) ‘0b100101’ >>> (37).bit_length() 6

**conjugate()**
   Returns self, the complex conjugate of any int.

**denominator**
   the denominator of a rational number in lowest terms

**imag**
   the imaginary part of a complex number

**numerator**
   the numerator of a rational number in lowest terms

**real**
   the real part of a complex number

class robot.reporting.stringcache.StringCache
   Bases: object
   add(text)
   dump()

**robot.reporting.xunitwriter module**

class robot.reporting.xunitwriter.XUnitWriter(**execution_result**, **skip_noncritical**)
   Bases: object
   write(output)

class robot.reporting.xunitwriter.XUnitFileWriter(**xml_writer**, **skip_noncritical=False**)
   Bases: robot.result.visitor.ResultVisitor
   Provides an xUnit-compatible result file.
   Attempts to adhere to the de facto schema guessed by Peter Reilly, see: http://marc.info/?l=ant-dev&m=123551933508682

**start_suite**(**suite**)  
   Called when suite starts. Default implementation does nothing.
   Can return explicit False to stop visiting.

**end_suite**(**suite**)  
   Called when suite ends. Default implementation does nothing.

**visit_test**(**test**)  
   Implements traversing through the test and its keywords.
   Can be overridden to allow modifying the passed in test without calling start_test() or end_test() nor visiting keywords.

**visit_keyword**(**kw**)  
   Implements traversing through the keyword and its child keywords.
   Can be overridden to allow modifying the passed in kw without calling start_keyword() or end_keyword() nor visiting child keywords.

**visit_statistics**(**stats**)  

**visit_errors**(**errors**)  

---

4.1. robot package 243
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.
Robot Framework Documentation, Release 3.2b3.dev1

visit_suite_statistics(stats)
visit_tag_statistics(stats)
visit_total_statistics(stats)

robot.result package

Implements parsing execution results from XML output files.

The main public API of this package consists of the ExecutionResult() factory method, that returns Result objects, and of the ResultVisitor abstract class, that eases further processing the results.

The model objects in the model module can also be considered to be part of the public API, because they can be found inside the Result object. They can also be inspected and modified as part of the normal test execution by pre-Rebot modifiers and listeners.

It is highly recommended to import the public entry-points via the robot.api package like in the example below. In those rare cases where the aforementioned model objects are needed directly, they can be imported from this package.

This package is considered stable.

Example

```python
#!/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__':
    (continues on next page)
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
based on them.

visit_suite (suite)

Implements traversing through the suite and its direct children.

Can be overridden to allow modifying the passed in suite without calling start_suite() or
end_suite() nor visiting child suites, tests or keywords (setup and teardown) at all.

add_tags

derived_keyword (keyword)

Called when keyword ends. Default implementation does nothing.

derived_message (msg)

Called when message ends. Default implementation does nothing.

derived_suite (suite)

Called when suite ends. Default implementation does nothing.

derived_test (test)

Called when test ends. Default implementation does nothing.

remove_tags

start_keyword (keyword)

Called when keyword starts. Default implementation does nothing.

Can return explicit False to stop visiting.

start_message (msg)

Called when message starts. Default implementation does nothing.

Can return explicit False to stop visiting.

start_suite (suite)

Called when suite starts. Default implementation does nothing.

Can return explicit False to stop visiting.
**start_test** *(test)*
Called when test starts. Default implementation does nothing.
Can return explicit False to stop visiting.

**visit_keyword** *(kw)*
Implements traversing through the keyword and its child keywords.
Can be overridden to allow modifying the passed in kw without calling start_keyword() or end_keyword() nor visiting child keywords.

**visit_message** *(msg)*
Implements visiting the message.
Can be overridden to allow modifying the passed in msg without calling start_message() or end_message().

**visit_test** *(test)*
Implements traversing through the test and its keywords.
Can be overridden to allow modifying the passed in test without calling start_test() or end_test() nor visiting keywords.

---

**robot.result.executionerrors module**

**class** robot.result.executionerrors.ExecutionErrors *(messages=None)*
Bases: object
Represents errors occurred during the execution of tests.
An error might be, for example, that importing a library has failed.

**message_class**
alias of robot.result.model.Message

**messages**
A list-like object of Message instances.

**add**(other)

**visit**(visitor)

---

**robot.result.executionresult module**

**class** robot.result.executionresult.Result *(source=None, root_suite=None, errors=None, rpa=None)*
Bases: object
Test execution results.
Can be created based on XML output files using the ExecutionResult() factory method. Also returned by the robot.running.TestSuite.run method.

**source = None**
Path to the XML file where results are read from.

**suite = None**
Hierarchical execution results as a TestSuite object.

**errors = None**
Execution errors as an ExecutionErrors object.
statistics
Test execution statistics.

Statistics are an instance of `Statistics` that is created based on the contained `suite` and possible `configuration`.

Statistics are created every time this property is accessed. Saving them to a variable is thus often a good idea to avoid re-creating them unnecessarily:

```python
from robot.api import ExecutionResult
result = ExecutionResult('output.xml')
result.configure(stat_config={'suite_stat_level': 2,
                              'tag_stat_combine': 'tagANDanother'})
stats = result.statistics
print stats.total.critical.failed
print stats.total.critical.passed
print stats.tags.combined[0].total
```

**return_code**
Return code (integer) of test execution.

By default returns the number of failed critical tests (max 250), but can be configured to always return 0.

**configure** *(status_rc=True, suite_config=None, stat_config=None)*
Configures the result object and objects it contains.

Parameters
- **status_rc** – If set to False, `return_code` always returns 0.
- **suite_config** – A dictionary of configuration options passed to `configure()` method of the contained `suite`.
- **stat_config** – A dictionary of configuration options used when creating `statistics`.

**save** *(path=None)*
Save results as a new output XML file.

Parameters **path** – Path to save results to. If omitted, overwrites the original file.

**visit** *(visitor)*
An entry point to visit the whole result object.

Parameters **visitor** – An instance of `ResultVisitor`.

Visitors can gather information, modify results, etc. See `result` package for a simple usage example.

Notice that it is also possible to call `result.suite.visit` if there is no need to visit the contained `statistics` or `errors`.

**handle_suite_teardown_failures** ()
Internal usage only.

**set_execution_mode** *(other)*
Set execution mode based on other result. Internal usage only.

**class** `robot.result.executionresult.CombinedResult` *(results=None)*
Bases: `robot.result.executionresult.Result`

Combined results of multiple test executions.
add_result (other)

configure (status_rc=True, suite_config=None, stat_config=None)
   Configures the result object and objects it contains.
   Parameters
   • status_rc – If set to False, return_code always returns 0.
   • suite_config – A dictionary of configuration options passed to configure() method of the contained suite.
   • stat_config – A dictionary of configuration options used when creating statistics.

handle_suite_teardown_failures ()
   Internal usage only.

return_code
   Return code (integer) of test execution.
   By default returns the number of failed critical tests (max 250), but can be configured to always return 0.

save (path=None)
   Save results as a new output XML file.
   Parameters path – Path to save results to. If omitted, overwrites the original file.

set_execution_mode (other)
   Set execution mode based on other result. Internal usage only.

statistics
   Test execution statistics.
   Statistics are an instance of Statistics that is created based on the contained suite and possible configuration.
   Statistics are created every time this property is accessed. Saving them to a variable is thus often a good idea to avoid re-creating them unnecessarily:

```
from robot.api import ExecutionResult
result = ExecutionResult('output.xml')
result.configure(stat_config={'suite_stat_level': 2, 'tag_stat_combine': 'tagANDanother'})
stats = result.statistics
print stats.total.critical.failed
print stats.total.critical.passed
print stats.tags.combined[0].total
```

visit (visitor)
   An entry point to visit the whole result object.
   Parameters visitor – An instance of ResultVisitor.
   Visitors can gather information, modify results, etc. See result package for a simple usage example.
   Notice that it is also possible to call result.suite.visit if there is no need to visit the contained statistics or errors.
robot.result.flattenkeywordmatcher module

robot.result.flattenkeywordmatcher.validate_flatten_keyword(options)

class robot.result.flattenkeywordmatcher.FlattenByTypeMatcher(flatten)
    Bases: object
    match(kwtype)

class robot.result.flattenkeywordmatcher.FlattenByNameMatcher(flatten)
    Bases: object
    match(kwname, libname=None)

class robot.result.flattenkeywordmatcher.FlattenByTagMatcher(flatten)
    Bases: object
    match(kwtags)

robot.result.keywordremover module

robot.result.keywordremover.KeywordRemover(how)

class robot.result.keywordremover.AllKeywordsRemover
    Bases: robot.result.keywordremover._KeywordRemover
    visit_keyword(keyword)
        Implements traversing through the keyword and its child keywords.
        Can be overridden to allow modifying the passed in kw without calling
        start_keyword() or end_keyword() nor visiting child keywords.

    end_keyword(keyword)
        Called when keyword ends. Default implementation does nothing.

    end_message(msg)
        Called when message ends. Default implementation does nothing.

    end_suite(suite)
        Called when suite ends. Default implementation does nothing.

    end_test(test)
        Called when test ends. Default implementation does nothing.

    start_keyword(keyword)
        Called when keyword starts. Default implementation does nothing.
        Can return explicit False to stop visiting.

    start_message(msg)
        Called when message starts. Default implementation does nothing.
        Can return explicit False to stop visiting.

    start_suite(suite)
        Called when suite starts. Default implementation does nothing.
        Can return explicit False to stop visiting.

    start_test(test)
        Called when test starts. Default implementation does nothing.
        Can return explicit False to stop visiting.
visit_message (msg)
Implements visiting the message.
Can be overridden to allow modifying the passed in msg without calling start_message() or end_message().

visit_suite (suite)
Implements traversing through the suite and its direct children.
Can be overridden to allow modifying the passed in suite without calling start_suite() or end_suite() nor visiting child suites, tests or keywords (setup and teardown) at all.

visit_test (test)
Implements traversing through the test and its keywords.
Can be overridden to allow modifying the passed in test without calling start_test() or end_test() nor visiting keywords.

class robot.result.keywordremover.PassedKeywordRemover
Bases: robot.result.keywordremover._KeywordRemover

start_suite (suite)
Called when suite starts. Default implementation does nothing.
Can return explicit False to stop visiting.

visit_test (test)
Implements traversing through the test and its keywords.
Can be overridden to allow modifying the passed in test without calling start_test() or end_test() nor visiting keywords.

visit_keyword (keyword)
Implements traversing through the keyword and its child keywords.
Can be overridden to allow modifying the passed in kw without calling start_keyword() or end_keyword() nor visiting child keywords.

end_keyword (keyword)
Called when keyword ends. Default implementation does nothing.

end_message (msg)
Called when message ends. Default implementation does nothing.

end_suite (suite)
Called when suite ends. Default implementation does nothing.

end_test (test)
Called when test ends. Default implementation does nothing.

start_keyword (keyword)
Called when keyword starts. Default implementation does nothing.
Can return explicit False to stop visiting.

start_message (msg)
Called when message starts. Default implementation does nothing.
Can return explicit False to stop visiting.

start_test (test)
Called when test starts. Default implementation does nothing.
Can return explicit False to stop visiting.
visit_message(msg)
Implements visiting the message.
Can be overridden to allow modifying the passed in msg without calling start_message() or end_message().

visit_suite(suite)
Implements traversing through the suite and its direct children.
Can be overridden to allow modifying the passed in suite without calling start_suite() or end_suite() nor visiting child suites, tests or keywords (setup and teardown) at all.

class robot.result.keywordremover.ByNameKeywordRemover(pattern)
Bases: robot.result.keywordremover._KeywordRemover

start_keyword(kw)
Called when keyword starts. Default implementation does nothing.
Can return explicit False to stop visiting.

end_keyword(keyword)
Called when keyword ends. Default implementation does nothing.

end_message(msg)
Called when message ends. Default implementation does nothing.

end_suite(suite)
Called when suite ends. Default implementation does nothing.

end_test(test)
Called when test ends. Default implementation does nothing.

start_message(msg)
Called when message starts. Default implementation does nothing.
Can return explicit False to stop visiting.

start_suite(suite)
Called when suite starts. Default implementation does nothing.
Can return explicit False to stop visiting.

start_test(test)
Called when test starts. Default implementation does nothing.
Can return explicit False to stop visiting.

visit_keyword(kw)
Implements traversing through the keyword and its child keywords.
Can be overridden to allow modifying the passed in kw without calling start_keyword() or end_keyword() nor visiting child keywords.

visit_message(msg)
Implements visiting the message.
Can be overridden to allow modifying the passed in msg without calling start_message() or end_message().

visit_suite(suite)
Implements traversing through the suite and its direct children.
Can be overridden to allow modifying the passed in suite without calling start_suite() or end_suite() nor visiting child suites, tests or keywords (setup and teardown) at all.
visit_test(test)
   Implements traversing through the test and its keywords.
   Can be overridden to allow modifying the passed in test without calling start_test() or end_test() nor visiting keywords.

class robot.result.keywordremover.ByTagKeywordRemover(pattern)
   Bases: robot.result.keywordremover._KeywordRemover

   start_keyword(kw)
      Called when keyword starts. Default implementation does nothing.
      Can return explicit False to stop visiting.

   end_keyword(keyword)
      Called when keyword ends. Default implementation does nothing.

   end_message(msg)
      Called when message ends. Default implementation does nothing.

   end_suite(suite)
      Called when suite ends. Default implementation does nothing.

   end_test(test)
      Called when test ends. Default implementation does nothing.

   start_message(msg)
      Called when message starts. Default implementation does nothing.
      Can return explicit False to stop visiting.

   start_suite(suite)
      Called when suite starts. Default implementation does nothing.
      Can return explicit False to stop visiting.

   start_test(test)
      Called when test starts. Default implementation does nothing.
      Can return explicit False to stop visiting.

   visit_keyword(kw)
      Implements traversing through the keyword and its child keywords.
      Can be overridden to allow modifying the passed in kw without calling start_keyword() or end_keyword() nor visiting child keywords.

   visit_message(msg)
      Implements visiting the message.
      Can be overridden to allow modifying the passed in msg without calling start_message() or end_message().

   visit_suite(suite)
      Implements traversing through the suite and its direct children.
      Can be overridden to allow modifying the passed in suite without calling start_suite() or end_suite() nor visiting child suites, tests or keywords (setup and teardown) at all.

   visit_test(test)
      Implements traversing through the test and its keywords.
      Can be overridden to allow modifying the passed in test without calling start_test() or end_test() nor visiting keywords.
```python
class robot.result.keywordremover.ForLoopItemsRemover
    Bases: robot.result.keywordremover._KeywordRemover

    start_keyword(kw)
        Called when keyword starts. Default implementation does nothing.
        Can return explicit False to stop visiting.

    end_keyword(keyword)
        Called when keyword ends. Default implementation does nothing.

    end_message(msg)
        Called when message ends. Default implementation does nothing.

    end_suite(suite)
        Called when suite ends. Default implementation does nothing.

    end_test(test)
        Called when test ends. Default implementation does nothing.

    start_message(msg)
        Called when message starts. Default implementation does nothing.
        Can return explicit False to stop visiting.

    start_suite(suite)
        Called when suite starts. Default implementation does nothing.
        Can return explicit False to stop visiting.

    start_test(test)
        Called when test starts. Default implementation does nothing.
        Can return explicit False to stop visiting.

    visit_keyword(kw)
        Implements traversing through the keyword and its child keywords.
        Can be overridden to allow modifying the passed in kw without calling start_keyword() or end_keyword() nor visiting child keywords.

    visit_message(msg)
        Implements visiting the message.
        Can be overridden to allow modifying the passed in msg without calling start_message() or end_message().

    visit_suite(suite)
        Implements traversing through the suite and its direct children.
        Can be overridden to allow modifying the passed in suite without calling start_suite() or end_suite() nor visiting child suites, tests or keywords (setup and teardown) at all.

    visit_test(test)
        Implements traversing through the test and its keywords.
        Can be overridden to allow modifying the passed in test without calling start_test() or end_test() nor visiting keywords.

class robot.result.keywordremover.WaitUntilKeywordSucceedsRemover
    Bases: robot.result.keywordremover._KeywordRemover

    start_keyword(kw)
        Called when keyword starts. Default implementation does nothing.
```

Can return explicit False to stop visiting.

**end_keyword** *(keyword)*
Called when keyword ends. Default implementation does nothing.

**end_message** *(msg)*
Called when message ends. Default implementation does nothing.

**end_suite** *(suite)*
Called when suite ends. Default implementation does nothing.

**end_test** *(test)*
Called when test ends. Default implementation does nothing.

**start_message** *(msg)*
Called when message starts. Default implementation does nothing.
Can return explicit False to stop visiting.

**start_suite** *(suite)*
Called when suite starts. Default implementation does nothing.
Can return explicit False to stop visiting.

**start_test** *(test)*
Called when test starts. Default implementation does nothing.
Can return explicit False to stop visiting.

**visit_keyword** *(kw)*
Implements traversing through the keyword and its child keywords.
Can be overridden to allow modifying the passed in kw without calling start_keyword() or end_keyword() nor visiting child keywords.

**visit_message** *(msg)*
Implements visiting the message.
Can be overridden to allow modifying the passed in msg without calling start_message() or end_message().

**visit_suite** *(suite)*
Implements traversing through the suite and its direct children.
Can be overridden to allow modifying the passed in suite without calling start_suite() or end_suite() nor visiting child suites, tests or keywords (setup and teardown) at all.

**visit_test** *(test)*
Implements traversing through the test and its keywords.
Can be overridden to allow modifying the passed in test without calling start_test() or end_test() nor visiting keywords.

**class** robot.result.keywordremover.WarningAndErrorFinder
**Bases:** robot.model.visitor.SuiteVisitor

**start_suite** *(suite)*
Called when suite starts. Default implementation does nothing.
Can return explicit False to stop visiting.

**start_test** *(test)*
Called when test starts. Default implementation does nothing.
Can return explicit False to stop visiting.
start_keyword(keyword)
Called when keyword starts. Default implementation does nothing.

Can return explicit False to stop visiting.

visit_message(msg)
Implements visiting the message.
Can be overridden to allow modifying the passed in msg without calling start_message() or end_message().

end_keyword(keyword)
Called when keyword ends. Default implementation does nothing.

end_message(msg)
Called when message ends. Default implementation does nothing.

end_suite(suite)
Called when suite ends. Default implementation does nothing.

end_test(test)
Called when test ends. Default implementation does nothing.

start_message(msg)
Called when message starts. Default implementation does nothing.

Can return explicit False to stop visiting.

visit_keyword(kw)
Implements traversing through the keyword and its child keywords.
Can be overridden to allow modifying the passed in kw without calling start_keyword() or end_keyword() nor visiting child keywords.

visit_suite(suite)
Implements traversing through the suite and its direct children.
Can be overridden to allow modifying the passed in suite without calling start_suite() or end_suite() nor visiting child suites, tests or keywords (setup and teardown) at all.

visit_test(test)
Implements traversing through the test and its keywords.
Can be overridden to allow modifying the passed in test without calling start_test() or end_test() nor visiting keywords.

class robot.result.keywordremover.RemovalMessage(message)
Bases: object

set_if_removed(kw, len_before)

set(kw, message=None)

robot.result.merger module

class robot.result.merger.Merger(result)
Bases: robot.model.visitor.SuiteVisitor

merge(merged)

start_suite(suite)
Called when suite starts. Default implementation does nothing.
Can return explicit `False` to stop visiting.

**end_suite** *(suite)*
Called when suite ends. Default implementation does nothing.

**visit_test** *(test)*
Implements traversing through the test and its keywords.
Can be overridden to allow modifying the passed in `test` without calling `start_test()` or `end_test()` nor visiting keywords.

**end_keyword** *(keyword)*
Called when keyword ends. Default implementation does nothing.

**end_message** *(msg)*
Called when message ends. Default implementation does nothing.

**end_test***(test)*
Called when test ends. Default implementation does nothing.

**start_keyword** *(keyword)*
Called when keyword starts. Default implementation does nothing.
Can return explicit `False` to stop visiting.

**start_message** *(msg)*
Called when message starts. Default implementation does nothing.
Can return explicit `False` to stop visiting.

**start_test** *(test)*
Called when test starts. Default implementation does nothing.
Can return explicit `False` to stop visiting.

**visit_keyword** *(kw)*
Implements traversing through the keyword and its child keywords.
Can be overridden to allow modifying the passed in `kw` without calling `start_keyword()` or `end_keyword()` nor visiting child keywords.

**visit_message** *(msg)*
Implements visiting the message.
Can be overridden to allow modifying the passed in `msg` without calling `start_message()` or `end_message()`.

**visit_suite** *(suite)*
Implements traversing through the suite and its direct children.
Can be overridden to allow modifying the passed in `suite` without calling `start_suite()` or `end_suite()` nor visiting child suites, tests or keywords (setup and teardown) at all.

**robot.result.messagefilter module**

```python
class robot.result.messagefilter.MessageFilter(loglevel):
    Bases: robot.model.visitor.SuiteVisitor

    start_keyword(keyword)
        Called when keyword starts. Default implementation does nothing.
        Can return explicit `False` to stop visiting.
```
**end_keyword** *(keyword)*
Called when keyword ends. Default implementation does nothing.

**end_message** *(msg)*
Called when message ends. Default implementation does nothing.

**end_suite** *(suite)*
Called when suite ends. Default implementation does nothing.

**end_test** *(test)*
Called when test ends. Default implementation does nothing.

**start_message** *(msg)*
Called when message starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

**start_suite** *(suite)*
Called when suite starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

**start_test** *(test)*
Called when test starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

**visit_keyword** *(kw)*
Implements traversing through the keyword and its child keywords.

Can be overridden to allow modifying the passed in `kw` without calling `start_keyword()` or `end_keyword()` nor visiting child keywords.

**visit_message** *(msg)*
Implements visiting the message.

Can be overridden to allow modifying the passed in `msg` without calling `start_message()` or `end_message()`.

**visit_suite** *(suite)*
Implements traversing through the suite and its direct children.

Can be overridden to allow modifying the passed in `suite` without calling `start_suite()` or `end_suite()` nor visiting child suites, tests or keywords (setup and teardown) at all.

**visit_test** *(test)*
Implements traversing through the test and its keywords.

Can be overridden to allow modifying the passed in `test` without calling `start_test()` or `end_test()` nor visiting keywords.

---

**robot.result.model module**

Module implementing result related model objects.

During test execution these objects are created internally by various runners. At that time they can inspected and modified by listeners.

When results are parsed from XML output files after execution to be able to create logs and reports, these objects are created by the `ExecutionResult()` factory method. At that point they can be inspected and modified by pre-Rebot modifiers.
The `ExecutionResult()` factory method can also be used by custom scripts and tools. In such usage it is often easiest to inspect and modify these objects using the `visitor interface`.

```python
class robot.result.model.Message(message=",  level='INFO',  html=False,  timestamp=None, parent=None)
    Bases: robot.model.message.Message
    
    Represents a single log message.
    
    See the base class for documentation of attributes not documented here.

copy(**attributes)
    Return shallow copy of this object.

    Parameters attributes – Attributes to be set for the returned copy automatically. For example, test.copy(name='New name').

    See also `deepcopy()`. The difference between these two is the same as with the standard `copy.copy` and `copy.deepcopy` functions that these methods also use internally.
    New in Robot Framework 3.0.1.

decopy(**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
    Returns the message content as HTML.

level
message
parent
timestamp
visit(visitor)
    Visitor interface entry-point.
```

```python
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.

deepe copy(**attributes)
   Return deep copy of this object.

   Parameters attributes -- Attributes to be set for the returned copy automatically. For example, test.deepe copy(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.
Keyword id in format like s1-t3-k1.

See TestSuite.id for more information.

```
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.

source

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.

4.1. robot package
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
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

source

tags
Test tags as a Tags object.

timeout

visit(visitor)
Visitor interface entry-point.

class robot.result.model.TestSuite(name=",", doc="", metadata=None, source=None, message="", starttime=None, endtime=None, rpa=False)

Bases: robot.model.testsuite.TestSuite

Represents results of a single test suite.

See the base class for documentation of attributes not documented here.

test_class
alias of TestCase

keyword_class
alias of Keyword

message
Possible suite setup or teardown error message.

starttime
Suite execution start time in format %Y%m%d %H:%M:%S.%f.
endtime
   Suite execution end time in format %Y%m%d %H:%M:%S.%f.

passed
   True if no critical test has failed, False otherwise.

status
   'PASS' if no critical test has failed, 'FAIL' otherwise.

statistics
   Suite statistics as a TotalStatistics object.
   Recreated every time this property is accessed, so saving the results to a variable and inspecting it is often a good idea:

       stats = suite.statistics
       print(stats.critical.failed)
       print(stats.all.total)
       print(stats.message)

full_message
   Combination of message and stat_message.

stat_message
   String representation of the statistics.

elapsedtime
   Total execution time in milliseconds.

criticality
   Used by tests to determine are they considered critical or not.
   Normally configured using --critical and --noncritical command line options. Can be set programmatically using set_criticality() of the root test suite.

set_criticality(critical_tags=None, non_critical_tags=None)
   Sets which tags are considered critical and which non-critical.
   Parameters
      • critical_tags – Tags or patterns considered critical. See the documentation of the --critical option for more details.
      • non_critical_tags – Tags or patterns considered non-critical. See the documentation of the --noncritical option for more details.
   Tags can be given as lists of strings or, when giving only one, as single strings. This information is used by tests to determine are they considered critical or not.
   Criticality can be set only to the root test suite.

remove_keywords(how)
   Remove keywords based on the given condition.
   Parameters how – What approach to use when removing keywords. Either ALL, PASSED, FOR, WUKS, or NAME:<pattern>.
   For more information about the possible values see the documentation of the --removekeywords command line option.

filter_messages(log_level='TRACE')
   Remove log messages below the specified log_level.
configure (**options)
A shortcut to configure a suite using one method call.

Can only be used with the root test suite.

**Parameters** options – Passed to SuiteConfigurer that will then set suite attributes, call filter(), etc. as needed.

Example:

```python
suite.configure(remove_keywords='PASSED',
                critical_tags='smoke',
                doc='Smoke test results.')
```

copy (**attributes)
Return shallow copy of this object.

**Parameters** attributes – Attributes to be set for the returned copy automatically. For example, test.copy (name='New name').

See also deepcopy(). The difference between these two is the same as with the standard copy.copy and copy.deepcopy functions that these methods also use internally.

New in Robot Framework 3.0.1.

deepcopy (**attributes)
Return deep copy of this object.

**Parameters** attributes – Attributes to be set for the returned copy automatically. For example, test.deepcopy (name='New name').

See also copy(). The difference between these two is the same as with the standard copy.copy and copy.deepcopy functions that these methods also use internally.

New in Robot Framework 3.0.1.

doc

filter (included_suites=None, included_tests=None, included_tags=None, excluded_tags=None)
Select test cases and remove others from this suite.

Parameters have the same semantics as --suite, --test, --include, and --exclude command line options. All of them can be given as a list of strings, or when selecting only one, as a single string.

Child suites that contain no tests after filtering are automatically removed.

Example:

```python
suite.filter(included_tests=['Test 1', '* Example'],
             included_tags='priority-1')
```

handle_suite_teardown_failures ()
Internal usage only.

has_tests

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_emptySuites `preserve_direct_children=False`
Removes all child suites not containing any tests, recursively.

rpa
set_tags `(add=None, remove=None, persist=False)`
Add and/or remove specified tags to the tests in this suite.

Parameters

- **add** – Tags to add as a list or, if adding only one, as a single string.
- **remove** – Tags to remove as a list or as a single string. Can be given as patterns where * and ? work as wildcards.
- **persist** – Add/remove specified tags also to new tests added to this suite in the future.

source

suites
Child suites as a `TestSuites` object.

test_count
Number of the tests in this suite, recursively.

tests
Tests as a `TestCases` object.

visit `visitor`
Visitor interface entry-point.

suite teardown failed `message`
Internal usage only.

**robot.result.resultbuilder module**

robot.result.resultbuilder `ExecutionResult` (*sources*, **options**)  
Factory method to constructs `Result` objects.

Parameters

- **sources** – XML source(s) containing execution results. Can be specified as paths, opened file objects, or strings/bytes containing XML directly. Support for bytes is new in RF 3.2.
- **options** – Configuration options. Using `merge=True` causes multiple results to be combined so that tests in the latter results replace the ones in the original. Setting `rpa` either to `True` (RPA mode) or `False` (test automation) sets execution mode explicitly. By default it is got from processed output files and conflicting modes cause an error. Other options are passed directly to the `ExecutionResultBuilder` object used internally.
Returns `Result` instance.

Should be imported by external code via the `robot.api` package. See the `robot.result` package for a usage example.

class robot.result.resultbuilder.ExecutionResultBuilder(source, include_keywords=True, flattened_keywords=None)

Bases: object

Builds `Result` objects based on output files.

Instead of using this builder directly, it is recommended to use the `ExecutionResult()` factory method.

Parameters

- `source` – Path to the XML output file to build `Result` objects from.
- `include_keywords` – Boolean controlling whether to include keyword information in the result or not. Keywords are not needed when generating only report.
- `flatten_keywords` – List of patterns controlling what keywords to flatten. See the documentation of `--flattenkeywords` option for more details.

build(result)

class robot.result.resultbuilder.RemoveKeywords

Bases: robot.model.visitor.SuiteVisitor

start_suite(suite)

Called when suite starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

visit_test(test)

Implements traversing through the test and its keywords.

Can be overridden to allow modifying the passed in `test` without calling `start_test()` or `end_test()` nor visiting keywords.

end_keyword(keyword)

Called when keyword ends. Default implementation does nothing.

end_message(msg)

Called when message ends. Default implementation does nothing.

end_suite(suite)

Called when suite ends. Default implementation does nothing.

end_test(test)

Called when test ends. Default implementation does nothing.

start_keyword(keyword)

Called when keyword starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_message(msg)

Called when message starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

start_test(test)

Called when test starts. Default implementation does nothing.
Can return explicit \texttt{False} to stop visiting.

\textbf{visit\_keyword}(kw)
Implementing 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.

\textbf{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()}.

\textbf{visit\_suite}(suite)
Implementing 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.

\textbf{robot.result.suiteteardownfailed module}\n
\begin{verbatim}
 class robot.result.suiteteardownfailed.SuiteTeardownFailureHandler
 Bases: robot.model.visitor.SuiteVisitor

 end_suite(suite)
 Called when suite ends. Default implementation does nothing.

 visit_test(test)
 Implements traversing through the test and its keywords.

 Can be overridden to allow modifying the passed in \texttt{test} without calling \texttt{start\_test()} or \texttt{end\_test()} nor visiting keywords.

 visit_keyword(keyword)
 Implements traversing through the keyword and its child keywords.

 Can be overridden to allow modifying the passed in \texttt{kw} without calling \texttt{start\_keyword()} or \texttt{end\_keyword()} nor visiting child keywords.

 end_keyword(keyword)
 Called when keyword ends. Default implementation does nothing.

 end_message(msg)
 Called when message ends. Default implementation does nothing.

 end_test(test)
 Called when test ends. Default implementation does nothing.

 start_keyword(keyword)
 Called when keyword starts. Default implementation does nothing.

 Can return explicit \texttt{False} to stop visiting.

 start_message(msg)
 Called when message starts. Default implementation does nothing.

 Can return explicit \texttt{False} to stop visiting.

 start_suite(suite)
 Called when suite starts. Default implementation does nothing.

 Can return explicit \texttt{False} to stop visiting.
\end{verbatim}
**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() or visiting child suites, tests or keywords (setup and teardown) at all.

class robot.result.suiteteardownfailed.SuiteTeardownFailed(error)
Bases: robot.model.visitor.SuiteVisitor

**visit_test**(test)
Implements traversing through the test and its keywords.
Can be overridden to allow modifying the passed in test without calling start_test() or end_test() nor visiting keywords.

**visit_keyword**(keyword)
Implements traversing through the keyword and its child keywords.
Can be overridden to allow modifying the passed in kw without calling start_keyword() or end_keyword() nor visiting child keywords.

**end_keyword**(keyword)
Called when keyword ends. Default implementation does nothing.

**end_message**(msg)
Called when message ends. Default implementation does nothing.

**end_suite**(suite)
Called when suite ends. Default implementation does nothing.

**end_test**(test)
Called when test ends. Default implementation does nothing.

**start_keyword**(keyword)
Called when keyword starts. Default implementation does nothing.
Can return explicit False to stop visiting.

**start_message**(msg)
Called when message starts. Default implementation does nothing.
Can return explicit False to stop visiting.

**start_suite**(suite)
Called when suite starts. Default implementation does nothing.
Can return explicit False to stop visiting.

**start_test**(test)
Called when test starts. Default implementation does nothing.
Can return explicit False to stop visiting.
visit_message (msg)
Implements visiting the message.
Can be overridden to allow modifying the passed in msg without calling start_message() or end_message().

visit_suite (suite)
Implements traversing through the suite and its direct children.
Can be overridden to allow modifying the passed in suite without calling start_suite() or end_suite() nor visiting child suites, tests or keywords (setup and teardown) at all.

robot.result.visitor module

Visitors can be used to easily traverse result structures.
This module contains ResultVisitor for traversing the whole Result object. It extends SuiteVisitor that contains visiting logic for the test suite structure.

class robot.result.visitor.ResultVisitor
    Bases: robot.model.visitor.SuiteVisitor
    Abstract class to conveniently travel Result objects.

A visitor implementation can be given to the visit() method of a result object. This will cause the result object to be traversed and the visitor’s visit_x(), start_x(), and end_x() methods to be called for each suite, test, keyword and message, as well as for errors, statistics, and other information in the result object. See methods below for a full list of available visitor methods.

See the result package level documentation for more information about handling results and a concrete visitor example. For more information about the visitor algorithm see documentation in robot.model.visitor module.

visit_result (result)
start_result (result)
end_result (result)
visit_statistics (stats)
start_statistics (stats)
end_statistics (stats)
visit_total_statistics (stats)
start_total_statistics (stats)
end_total_statistics (stats)
visit_tag_statistics (stats)
start_tag_statistics (stats)
end_tag_statistics (stats)
visit_suite_statistics (stats)
start_suite_statistics (stats)
end_suite_statistics (suite_stats)
visit_stat (stat)
start_stat(stat)
end_stat(stat)

visit_errors(errors)
start_errors(errors)
end_errors(errors)

end_keyword(keyword)
   Called when keyword ends. Default implementation does nothing.

end_message(msg)
   Called when message ends. Default implementation does nothing.

end_suite(suite)
   Called when suite ends. Default implementation does nothing.

end_test(test)
   Called when test ends. Default implementation does nothing.

start_keyword(keyword)
   Called when keyword starts. Default implementation does nothing.

Can return explicit False to stop visiting.

start_message(msg)
   Called when message starts. Default implementation does nothing.

Can return explicit False to stop visiting.

start_suite(suite)
   Called when suite starts. Default implementation does nothing.

Can return explicit False to stop visiting.

start_test(test)
   Called when test starts. Default implementation does nothing.

Can return explicit False to stop visiting.

visit_keyword(kw)
   Implements traversing through the keyword and its child keywords.

Can be overridden to allow modifying the passed in kw without calling start_keyword() or end_keyword() nor visiting child keywords.

visit_message(msg)
   Implements visiting the message.

Can be overridden to allow modifying the passed in msg without calling start_message() or end_message().

visit_suite(suite)
   Implements traversing through the suite and its direct children.

Can be overridden to allow modifying the passed in suite without calling start_suite() or end_suite() nor visiting child suites, tests or keywords (setup and teardown) at all.

visit_test(test)
   Implements traversing through the test and its keywords.

Can be overridden to allow modifying the passed in test without calling start_test() or end_test() nor visiting keywords.
robot.result.xmlelementhandlers module

class robot.result.xmlelementhandlers.XmlElementHandler (execution_result, root_handler=None)
        Bases: object
        start (elem)
        end (elem)

class robot.result.xmlelementhandlers.RootHandler
        Bases: robot.result.xmlelementhandlers._Handler
        end (elem, result)
        get_child_handler (elem)
        start (elem, result)

class robot.result.xmlelementhandlers.RobotHandler
        Bases: robot.result.xmlelementhandlers._Handler
        tag = 'robot'
        start (elem, result)
        end (elem, result)
        get_child_handler (elem)

class robot.result.xmlelementhandlers.SuiteHandler
        Bases: robot.result.xmlelementhandlers._Handler
        tag = 'suite'
        start (elem, result)
        end (elem, result)
        get_child_handler (elem)

class robot.result.xmlelementhandlers.RootSuiteHandler
        Bases: robot.result.xmlelementhandlers.SuiteHandler
        start (elem, result)
        end (elem, result)
        get_child_handler (elem)
        tag = 'suite'

class robot.result.xmlelementhandlers.TestCaseHandler
        Bases: robot.result.xmlelementhandlers._Handler
        tag = 'test'
        start (elem, result)
        end (elem, result)
        get_child_handler (elem)

class robot.result.xmlelementhandlers.KeywordHandler
        Bases: robot.result.xmlelementhandlers._Handler
        tag = 'kw'
```python
start (elem, result)
end (elem, result)
get_child_handler (elem)

class robot.result.xmlelementhandlers.MessageHandler
    Bases: robot.result.xmlelementhandlers._Handler
    tag = 'msg'
end (elem, result)
get_child_handler (elem)
start (elem, result)

class robot.result.xmlelementhandlers.KeywordStatusHandler
    Bases: robot.result.xmlelementhandlers._StatusHandler
    end (elem, result)
    get_child_handler (elem)
    start (elem, result)
    tag = 'status'

class robot.result.xmlelementhandlers.SuiteStatusHandler
    Bases: robot.result.xmlelementhandlers._StatusHandler
    end (elem, result)
    get_child_handler (elem)
    start (elem, result)
    tag = 'status'

class robot.result.xmlelementhandlers.TestStatusHandler
    Bases: robot.result.xmlelementhandlers._StatusHandler
    end (elem, result)
    get_child_handler (elem)
    start (elem, result)
    tag = 'status'

class robot.result.xmlelementhandlers.DocHandler
    Bases: robot.result.xmlelementhandlers._Handler
    tag = 'doc'
end (elem, result)
get_child_handler (elem)
start (elem, result)

class robot.result.xmlelementhandlers.MetadataHandler
    Bases: robot.result.xmlelementhandlers._Handler
    tag = 'metadata'
end (elem, result)
get_child_handler (elem)
```
start (elem, result)
class robot.result.xmlelementhandlers.MetadataItemHandler
    Bases: robot.result.xmlelementhandlers._Handler
    tag = 'item'
    end (elem, result)
    get_child_handler (elem)
    start (elem, result)
class robot.result.xmlelementhandlers.TagsHandler
    Bases: robot.result.xmlelementhandlers._Handler
    tag = 'tags'
    end (elem, result)
    get_child_handler (elem)
    start (elem, result)
class robot.result.xmlelementhandlers.TagHandler
    Bases: robot.result.xmlelementhandlers._Handler
    tag = 'tag'
    end (elem, result)
    get_child_handler (elem)
    start (elem, result)
class robot.result.xmlelementhandlers.TimeoutHandler
    Bases: robot.result.xmlelementhandlers._Handler
    tag = 'timeout'
    end (elem, result)
    get_child_handler (elem)
    start (elem, result)
class robot.result.xmlelementhandlers.AssignHandler
    Bases: robot.result.xmlelementhandlers._Handler
    tag = 'assign'
    end (elem, result)
    get_child_handler (elem)
    start (elem, result)
class robot.result.xmlelementhandlers.AssignVarHandler
    Bases: robot.result.xmlelementhandlers._Handler
    tag = 'var'
    end (elem, result)
    get_child_handler (elem)
    start (elem, result)
class robot.result.xmlelementhandlers.ArgumentsHandler
    Bases: robot.result.xmlelementhandlers._Handler
    tag = 'arguments'
    end(elem, result)
    get_child_handler(elem)
    start(elem, result)

class robot.result.xmlelementhandlers.ArgumentHandler
    Bases: robot.result.xmlelementhandlers._Handler
    tag = 'arg'
    end(elem, result)
    get_child_handler(elem)
    start(elem, result)

class robot.result.xmlelementhandlers.ErrorsHandler
    Bases: robot.result.xmlelementhandlers._Handler
    tag = 'errors'
    start(elem, result)
    end(elem, result)
    get_child_handler(elem)

class robot.result.xmlelementhandlers.StatisticsHandler
    Bases: robot.result.xmlelementhandlers._Handler
    tag = 'statistics'
    get_child_handler(elem)
    end(elem, result)
    start(elem, result)

robot.running package

Implements the core test execution logic.

The main public entry points of this package are of the following two classes:

- **TestSuiteBuilder** for creating executable test suites based on existing test case files and directories.
- **TestSuite** for creating an executable test suite structure programmatically.

It is recommended to import both of these classes via the robot.api package like in the examples below. Also **TestCase** and **Keyword** classes used internally by the **TestSuite** class are part of the public API. In those rare cases where these classes are needed directly, they can be imported from this package.

Examples

First, let’s assume we have the following test suite in file activate_skynet.robot:
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:

```python
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

4.1. robot package 275
robot.running.arguments package

Submodules

robot.running.arguments.argumentconverter module

class robot.running.arguments.argumentconverter.ArgumentConverter(argspec, dry_run=False)

    Bases: object

    convert (positional, named)

robot.running.arguments.argumentmapper module

class robot.running.arguments.argumentmapper.ArgumentMapper(argspec)

    Bases: object

    map (positional, named, replace_defaults=True)

class robot.running.arguments.argumentmapper.KeywordCallTemplate(argspec)

    Bases: object

    fill_positional (positional)
    fill_named (named)
    replace_defaults ()

class robot.running.arguments.argumentmapper.DefaultValue(value)

    Bases: object

    resolve (variables)

robot.running.arguments.argumentparser module

robot.running.arguments.argumentparser.getfullargspec(func)

robot.running.arguments.argumentparser.unwrap(func)

class robot.running.arguments.argumentparser.PythonArgumentParser(type='Keyword')

    Bases: robot.running.arguments.argumentparser._ArgumentParser

    parse (handler, name=None)

class robot.running.arguments.argumentparser.JavaArgumentParser(type='Keyword')

    Bases: robot.running.arguments.argumentparser._ArgumentParser

    parse (signatures, name=None)

class robot.running.arguments.argumentparser.DynamicArgumentParser(type='Keyword')

    Bases: robot.running.arguments.argumentparser._ArgumentSpecParser

    parse (argspec, name=None)
class robot.running.arguments.argumentparser.UserKeywordArgumentParser(type='Keyword')
    Bases: robot.running.arguments.argumentparser._ArgumentSpecParser
    parse(argspec, name=None)

robot.running.arguments.argumentresolver module

class robot.running.arguments.argumentresolver.ArgumentParser(argspec, resolve_named=True, resolve_variables_until=None, dict_to_kwargs=False)
    Bases: object
    resolve(arguments, variables=None)

class robot.running.arguments.argumentresolver.NamedArgumentResolver(argspec)
    Bases: object
    resolve(arguments, variables=None)

class robot.running.arguments.argumentresolver.NullNamedArgumentResolver
    Bases: object
    resolve(arguments, variables=None)

class robot.running.arguments.argumentresolver.DictToKwargs(argspec, enabled=False)
    Bases: object
    handle(positional, named)

class robot.running.arguments.argumentresolver.VariableReplacer(resolve_until=None)
    Bases: object
    replace(positional, named, variables=None)

robot.running.arguments.argmentspec module

class robot.running.arguments.argmentspec.ArgumentSpec(name=None, type='Keyword', positional=None, varargs=None, kwonlyargs=None, kwargs=None, defaults=None, types=None, supports_named=True)
    Bases: object
    types
    minargs
    maxargs
    argument_names
    resolve(arguments, variables=None, resolve_named=True, resolve_variables_until=None, dict_to_kwargs=False)
    map(positional, named, replace_defaults=True)
robot.running.arguments.argumentvalidator module

class robot.running.arguments.argumentvalidator.ArgumentParser(argspec)
    Bases: object

    validate (positional, named, dryrun=False)

robot.running.arguments.embedded module

class robot.running.arguments.embedded.EmbeddedArguments(name)
    Bases: object

class robot.running.arguments.embedded.EmbeddedArgumentParser
    Bases: object

    parse(string)

robot.running.arguments.javaargumentcoercer module

robot.running.arguments.typeconverters module

class robot.running.arguments.typeconverters.Enum
    Bases: object

class robot.running.arguments.typeconverters.TypeConverter
    Bases: object

    type = None
    abc = None
    aliases = ()
    convert_none = True
    type_name
    classmethod register(converter_class)
    classmethod converter_for(type_)
    handles(type_)
    get_converter(type_)
    convert(name, value, explicit_type=True)

class robot.running.arguments.typeconverters.BooleanConverter
    Bases: robot.running.arguments.typeconverters.TypeConverter

    type
        alias of __builtin__.bool

    type_name = 'boolean'
    aliases = ('bool',)
    abc = None
    convert(name, value, explicit_type=True)
convert_none = True
classmethod converter_for(type_)
get_converter (type_)
handles (type_)
classmethod register (converter_class)

class robot.running.arguments.typeconverters.IntegerConverter
Bases: robot.running.arguments.typeconverters.TypeConverter

type
    alias of __builtin__.int

abc
    alias of numbers.Integral
type_name = 'integer'
aliases = ('int', 'long')
convert (name, value, explicit_type=True)
convert_none = True
classmethod converter_for(type_)
get_converter (type_)
handles (type_)
classmethod register (converter_class)

class robot.running.arguments.typeconverters.FloatConverter
Bases: robot.running.arguments.typeconverters.TypeConverter

type
    alias of __builtin__.float

abc
    alias of numbers.Real
aliases = ('double',)
convert (name, value, explicit_type=True)
convert_none = True
classmethod converter_for(type_)
get_converter (type_)
handles (type_)
classmethod register (converter_class)

type_name

class robot.running.arguments.typeconverters.DecimalConverter
Bases: robot.running.arguments.typeconverters.TypeConverter

type
    alias of decimal.Decimal

abc = None
aliases = ()
convert (name, value, explicit_type=True)
convert_none = True
class method converter_for (type_)
get_converter (type_)
handles (type_)
class method register (converter_class)
type_name

class robot.running.arguments.typeconverters.BytesConverter
Bases: robot.running.arguments.typeconverters.TypeConverter
type
    alias of __builtin__.str
abc = None
type_name = 'bytes'
convert_none = False
aliases = ()
convert (name, value, explicit_type=True)
class method converter_for (type_)
get_converter (type_)
handles (type_)
class method register (converter_class)

class robot.running.arguments.typeconverters.ByteArrayConverter
Bases: robot.running.arguments.typeconverters.TypeConverter
type
    alias of __builtin__.bytearray
convert_none = False
abc = None
aliases = ()
convert (name, value, explicit_type=True)
class method converter_for (type_)
get_converter (type_)
handles (type_)
class method register (converter_class)

class robot.running.arguments.typeconverters.DateTimeConverter
Bases: robot.running.arguments.typeconverters.TypeConverter
type
    alias of datetime.datetime
abc = None
aliases = ()
convert (name, value, explicit_type=True)
convert_none = True
classmethod converter_for(type_)
get_converter (type_)
handles (type_)
classmethod register (converter_class)
type_name
class robot.running.arguments.typeconverters.DateConverter
Bases: robot.running.arguments.typeconverters.TypeConverter
type
    alias of datetime.date
abc = None
aliases = ()
convert (name, value, explicit_type=True)
convert_none = True
classmethod converter_for(type_)
get_converter (type_)
handles (type_)
classmethod register (converter_class)
type_name
class robot.running.arguments.typeconverters.TimeDeltaConverter
Bases: robot.running.arguments.typeconverters.TypeConverter
type
    alias of datetime.timedelta
abc = None
aliases = ()
convert (name, value, explicit_type=True)
convert_none = True
classmethod converter_for(type_)
get_converter (type_)
handles (type_)
classmethod register (converter_class)
type_name
class robot.running.arguments.typeconverters.EnumConverter (enum=None)
Bases: robot.running.arguments.typeconverters.TypeConverter
type
    alias of Enum

4.1. robot package
```python
type_name
get_converter(type_)
abc = None
aliases = ()
convert(name, value, explicit_type=True)
convert_none = True
classmethod converter_for(type_)
handles(type_)
classmethod register(converter_class)
class robot.running.arguments.typeconverters.NoneConverter
    Bases: robot.running.arguments.typeconverters.TypeConverter
type
    alias of __builtin__.NoneType
abc = None
aliases = ()
convert(name, value, explicit_type=True)
convert_none = True
classmethod converter_for(type_)
get_converter(type_)
handles(type_)
classmethod register(converter_class)
type_name
class robot.running.arguments.typeconverters.ListConverter
    Bases: robot.running.arguments.typeconverters.TypeConverter
type
    alias of __builtin__.list
abc
    alias of _abcoll.Sequence
aliases = ()
convert(name, value, explicit_type=True)
convert_none = True
classmethod converter_for(type_)
get_converter(type_)
handles(type_)
classmethod register(converter_class)
type_name
class robot.running.arguments.typeconverters.TupleConverter
    Bases: robot.running.arguments.typeconverters.TypeConverter
```
type
    alias of __builtin__.tuple

abc = None
aliases = ()
convert (name, value, explicit_type=True)
convert_none = True
classmethod converter_for (type_)
get_converter (type_)
handles (type_)
classmethod register (converter_class)

type_name

class robot.running.arguments.typeconverters.DictionaryConverter
    Bases: robot.running.arguments.typeconverters.TypeConverter

type
    alias of __builtin__.dict
abc
    alias of _abcoll.Mapping
type_name = 'dictionary'
aliases = ('dict', 'map')
convert (name, value, explicit_type=True)
convert_none = True
classmethod converter_for (type_)
get_converter (type_)
handles (type_)
classmethod register (converter_class)

class robot.running.arguments.typeconverters.SetConverter
    Bases: robot.running.arguments.typeconverters.TypeConverter

type
    alias of __builtin__.set
abc
    alias of _abcoll.Set
aliases = ()
convert (name, value, explicit_type=True)
convert_none = True
classmethod converter_for (type_)
get_converter (type_)
handles (type_)
classmethod register (converter_class)
class robot.running.arguments.typeconverters.FrozenSetConverter
    Bases: robot.running.arguments.typeconverters.TypeConverter

    type
        alias of __builtin__.frozenset

    abc = None
    aliases = ()

    convert(name, value, explicit_type=True)
    convert_none = True

    classmethod converter_for(type_)

    get_converter(type_)

    handles(type_)

    classmethod register(converter_class)

robot.running.arguments.typevalidator module

class robot.running.arguments.typevalidator.TypeValidator(argspec)
    Bases: object

    validate(types)

    validate_type_dict(types)

    convert_type_list_to_dict(types)

robot.running.builder package

Submodules

robot.running.builder.builders module

class robot.running.builder.builders.TestSuiteBuilder(included_suites=None, included_extensions=('robot', ), rpa=None, allow_empty_suite=False, process_curdir=True)

    Bases: object

    build(*paths)

        Parameters paths -- Paths to test data files or directories.

        Returns TestSuite instance.
class robot.running.builder.builders.SuiteStructureParser(included_extensions,
    rpa=None,
    process_curdir=True)
    Bases: robot.parsing.suitestructure.SuiteStructureVisitor
    parse(structure)
    visit_file(structure)
    start_directory(structure)
    end_directory(structure)
    visit_directory(structure)

class robot.running.builder.builders.ResourceFileBuilder(process_curdir=True)
    Bases: object
    build(source)

robot.running.builder.parsers module

class robot.running.builder.parsers.BaseParser
    Bases: object
    parse_init_file(source, defaults=None)
    parse_suite_file(source, defaults=None)
    parse_resource_file(source)

class robot.running.builder.parsers.RobotParser(process_curdir=True)
    Bases: robot.running.builder.parsers.BaseParser
    parse_init_file(source, defaults=None)
    parse_suite_file(source, defaults=None)
    build_suite(model, name=None, defaults=None)
    parse_resource_file(source)

class robot.running.builder.parsers.RestParser(process_curdir=True)
    Bases: robot.running.builder.parsers.RobotParser
    build_suite(model, name=None, defaults=None)
    parse_init_file(source, defaults=None)
    parse_resource_file(source)
    parse_suite_file(source, defaults=None)

class robot.running.builder.parsers.NoInitFileDirectoryParser
    Bases: robot.running.builder.parsers.BaseParser
    parse_init_file(source, defaults=None)
    parse_resource_file(source)
    parse_suite_file(source, defaults=None)

robot.running.builder.parsers.format_name(source)

class robot.running.builder.parsers.ErrorReporter(source)
    Bases: ast.NodeVisitor

4.1. robot package
visit_Error(node)

generic_visit(node)
   Called if no explicit visitor function exists for a node.

visit(node)
   Visit a node.

**robot.running.builder.testsettings module**

class robot.running.builder.testsettings.TestDefaults(parent=None)
    Bases: object
    setup
tear down
force_tags
timeout

class robot.running.builder.testsettings.TestSettings(defaults)
    Bases: object
    setup
tear down
timeout
template
tags

**robot.running.builder.transformers module**

robot.running.builder.transformers.fixture(node, fixture_type)

class robot.running.builder.transformers.SettingsBuilder(suite, test_defaults)
    Bases: ast.NodeVisitor
    visit_Documentation(node)
    visit_Metadata(node)
    visit_SuiteSetup(node)
    visit_SuiteTeardown(node)
    visit_TestSetup(node)
    visit_TestTeardown(node)
    visit_TestTimeout(node)
    visit_DefaultTags(node)
    visit_ForceTags(node)
    visit_TestTemplate(node)
    visit_ResourceImport(node)
    visit_LibraryImport(node)
visit_VariablesImport (node)
visit_VariableSection (node)
visit_TestCaseSection (node)
visit_KeywordSection (node)
generic_visit (node)
    Called if no explicit visitor function exists for a node.
visit (node)
    Visit a node.

class robot.running.builder.transformers.SuiteBuilder (suite, test_defaults)
    Bases: ast.NodeVisitor
    visit_SettingSection (node)
    visit_Variable (node)
    visit_TestCase (node)
    visit_Keyword (node)
generic_visit (node)
    Called if no explicit visitor function exists for a node.
visit (node)
    Visit a node.

class robot.running.builder.transformers.ResourceBuilder (resource)
    Bases: ast.NodeVisitor
    visit_Documentation (node)
    visit_LibraryImport (node)
    visit_ResourceImport (node)
    visit_VariablesImport (node)
    visit_Variable (node)
    visit_Keyword (node)
generic_visit (node)
    Called if no explicit visitor function exists for a node.
visit (node)
    Visit a node.

class robot.running.builder.transformers.TestCaseBuilder (suite, defaults)
    Bases: ast.NodeVisitor
    visit_TestCase (node)
    visit_ForLoop (node)
    visit_TemplateArguments (node)
    visit_Documentation (node)
    visit_Setup (node)
    visit_Teardown (node)
    visit_Timeout (node)
visit_Tags(node)
visit_Template(node)
visit_KeywordCall(node)
generic_visit(node)
    Called if no explicit visitor function exists for a node.

visit(node)
    Visit a node.

class robot.running.builder.transformers.KeywordBuilder(resource)
    Bases: ast.NodeVisitor
visit_Keyword(node)
visit_Documentation(node)
visit_Arguments(node)
visit_Tags(node)
visit_Return(node)
visit_Timeout(node)
visit_Teardown(node)
visit_KeywordCall(node)
visit_ForLoop(node)
generic_visit(node)
    Called if no explicit visitor function exists for a node.

visit(node)
    Visit a node.

class robot.running.builder.transformers.ForLoopBuilder(loop)
    Bases: ast.NodeVisitor
visit_KeywordCall(node)
visit_TemplateArguments(node)
generic_visit(node)
    Called if no explicit visitor function exists for a node.

visit(node)
    Visit a node.

robot.running.timeouts package

class robot.running.timeouts.TestTimeout(timeout=None, variables=None, rpa=False)
    Bases: robot.running.timeouts._Timeout
    type = 'Test'
    set_keyword_timeout(timeout_occurred)
    any_timeout_occurred()
    active
    get_message()
replace_variables(variables)
run(runnable, args=None, kwargs=None)
start()
time_left()
timed_out()

class robot.running.timeouts.KeywordTimeout(timeout=None, variables=None)
    Bases: robot.running.timeouts._Timeout
    active
    get_message()
    replace_variables(variables)
    run(runnable, args=None, kwargs=None)
    start()
time_left()
timed_out()
type = 'Keyword'

Submodules

robot.running.timeouts.ironpython module

robot.running.timeouts.jython module

robot.running.timeouts.posix module

class robot.running.timeoutsposix.Timeout(timeout, error)
    Bases: object
    execute(runnable)

robot.running.timeouts.windows module

class robot.running.timeouts.windows.Timeout(timeout, error)
    Bases: object
    execute(runnable)

Submodules

robot.running.context module

class robot.running.context.ExecutionContexts
    Bases: object
    current
top

namespaces

start_suite (suite, namespace, output, dry_run=False)

end_suite()

robot.running.dynamicmethods module

robot.running.dynamicmethods.no_dynamic_method(*args)

class robot.running.dynamicmethods.GetKeywordNames(lib)
    Bases: robot.running.dynamicmethods._DynamicMethod
        name

class robot.running.dynamicmethods.RunKeyword(lib)
    Bases: robot.running.dynamicmethods._DynamicMethod
        supports_kwargs
        name

class robot.running.dynamicmethods.GetKeywordDocumentation(lib)
    Bases: robot.running.dynamicmethods._DynamicMethod
        name

class robot.running.dynamicmethods.GetKeywordArguments(lib)
    Bases: robot.running.dynamicmethods._DynamicMethod
        name

class robot.running.dynamicmethods.GetKeywordTypes(lib)
    Bases: robot.running.dynamicmethods._DynamicMethod
        name

class robot.running.dynamicmethods.GetKeywordTags(lib)
    Bases: robot.running.dynamicmethods._DynamicMethod
        name

robot.running.handlers module

robot.running.handlers.Handler(library, name, method)
robot.running.handlers.DynamicHandler(library, name, method, doc, argspec, tags=None)
robot.running.handlers.InitHandler(library, method, docgetter=None)

class robot.running.handlers.EmbeddedArgumentsHandler(name_regexp, orig_handler)
    Bases: object
        library
        matches(name)
        create_runner(name)
robot.running.handlerstore module

class robot.running.handlerstore.HandlerStore(source, source_type)
    Bases: object
    TEST_LIBRARY_TYPE = 'Test library'
    TEST_CASE_FILE_TYPE = 'Test case file'
    RESOURCE_FILE_TYPE = 'Resource file'
    add(handler, embedded=False)
    create_runner(name)

robot.running.importer module

class robot.running.importer.Importer
    Bases: object
    reset()
    close_global_library_listeners()
    import_library(name, args, alias, variables)
    import_resource(path)

class robot.running.importer.ImportCache
    Bases: object
    Keeps track on and optionally caches imported items.
    Handles paths in keys case-insensitively on case-insensitive OSes. Unlike dicts, this storage accepts mutable values in keys.
    add(key, item=None)
    values()

robot.running.librarykeywordrunner module

class robot.running.librarykeywordrunner.LibraryKeywordRunner(handler, name=None)
    Bases: object
    library
    libname
    longname
    run(kw, context)
    dry_run(kw, context)

class robot.running.librarykeywordrunner.EmbeddedArgumentsRunner(handler, name)
    Bases: robot.running.librarykeywordrunner.LibraryKeywordRunner
    dry_run(kw, context)
    libname
```python
library
longname
run(kw, context)

class robot.running.librarykeywordrunner.RunKeywordRunner(handler, 
default_dry_run_keywords=False)
Bases: robot.running.librarykeywordrunner.LibraryKeywordRunner
dry_run(kw, context)
libname
library
longname
run(kw, context)

robot.running.libraryscopes module

robot.running.libraryscopes.LibraryScope(libcode, library)
class robot.running.libraryscopes.GlobalScope(library)
    Bases: object
        is_global = True
        start_suite()
        end_suite()
        start_test()
        end_test()
class robot.running.libraryscopes.TestSuiteScope(library)
    Bases: robot.running.libraryscopes.GlobalScope
        is_global
        start_suite()
        end_suite()
        end_test()
        start_test()
class robot.running.libraryscopes.TestCaseScope(library)
    Bases: robot.running.libraryscopes.TestSuiteScope
        start_test()
        end_test()
        end_suite()
        is_global
        start_suite()
```

292 Chapter 4. All packages
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', lineno=0)

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.

lineno

run(context)

Execute the keyword.

Typically called internally by TestSuite.run().

FOR_ITEM_TYPE = 'foritem'

FOR_LOOP_TYPE = 'for'

KEYWORD_TYPE = 'kw'

SETUP_TYPE = 'setup'

TEARDOWN_TYPE = 'teardown'

args

assign

children

Child keywords and messages in creation order.

copy(**attributes)

Return shallow copy of this object.

Parameters attributes – Attributes to be set for the returned copy automatically. For example, test.copy(name='New name').

See also deepcopy(). The difference between these two is the same as with the standard copy.copy and copy.deepcopy functions that these methods also use internally.

New in Robot Framework 3.0.1.

deepcopy(**attributes)

Return deep copy of this object.
Parameters attributes – Attributes to be set for the returned copy automatically. For example, `test.deepcopy(name='New name')`.

See also `copy()`. The difference between these two is the same as with the standard `copy.copy` and `copy.deepcopy` functions that these methods also use internally.

New in Robot Framework 3.0.1.

doc
id
    Keyword id in format like s1-t3-k1.

    See `TestSuite.id` for more information.

keyword_class = None
keywords
    Child keywords as a `Keywords` object.
messages
    Messages as a `Messages` object.
name
parent
    Parent test suite, test case or keyword.
source
tags
    Keyword tags as a `Tags` object.
timeout
type
visit(visitor)
    Visitor interface entry-point.

class robot.running.model.ForLoop(variables, values, flavor, lineno=None, _header='FOR', _end='END')

    Bases: `robot.running.model.Keyword`

    Represents a for loop in test data.
    Contains keywords in the loop body as child `keywords`.

keyword_class
    Internal usage only.

    alias of `Keyword`

flavor
lineno
variables
values
    FOR_ITEM_TYPE = 'foritem'
    FOR_LOOP_TYPE = 'for'
    KEYWORD_TYPE = 'kw'
    SETUP_TYPE = 'setup'
TEARDOWN_TYPE = 'teardown'

args
assign
children
    Child *keywords* and *messages* in creation order.
copy(**attributes**)
    Return shallow copy of this object.
    **Parameters attributes** – Attributes to be set for the returned copy automatically. For example, `test.copy(name='New name')`.
    See also `deepcopy()`. The difference between these two is the same as with the standard `copy.copy` and `copy.deepcopy` functions that these methods also use internally.
    New in Robot Framework 3.0.1.
deePCopy(**attributes**)
    Return deep copy of this object.
    **Parameters attributes** – Attributes to be set for the returned copy automatically. For example, `test.deepcopy(name='New name')`.
    See also `copy()`. The difference between these two is the same as with the standard `copy.copy` and `copy.deepcopy` functions that these methods also use internally.
    New in Robot Framework 3.0.1.
doc
id
    Keyword id in format like s1-t3-k1.
    See *TestSuite.id* for more information.
keywords
    Child keywords as a *Keywords* object.
message_class = None
messages
    Messages as a *Messages* object.
name
parent
    Parent test suite, test case or keyword.
run(*context*)
    Execute the keyword.
    Typically called internally by *TestSuite.run()*.
source
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, lineno=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 template used.

**lineno**

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.

depthcopy (**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.

levelname

Test name prefixed with the long name of the parent suite.

name

parent

source

tags

Test tags as a Tags object.

timeout
visit (visitor)

Visitor interface entry-point.

class robot.running.model.TestSuite (name=", doc=", metadata=None, source=None, rpa=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.

classmethod from_file_system (*paths, **config)

Create a TestSuite object based on the given paths.

paths are file or directory paths where to read the data from.

Internally utilizes the TestSuiteBuilder class and config can be used to configure how it is initialized.

New in Robot Framework 3.2.

classmethod from_model (model, name=None)

Create a TestSuite object based on the given model.

The model can be created by using the get_model() function and possibly modified by other tooling in the parsing module.

New in Robot Framework 3.2.

configure (randomize_suites=False, randomize_tests=False, randomize_seed=None, **options)

A shortcut to configure a suite using one method call.

Can only be used with the root test suite.

Parameters

• randomize_xxx – Passed to randomize().

• options – Passed to SuiteConfigurer that will then set suite attributes, call filter(), etc. as needed.

Example:

```python
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.

```python
run (settings=None, **options)
```

Executes the suite based based the given settings or options.

**Parameters**

- **settings** – RobotSettings object to configure test execution.
- **options** – Used to construct new RobotSettings object if settings are not given.

**Returns** Result object with information about executed suites and tests.

If options are used, their names are the same as long command line options except without hyphens. Some options are ignored (see below), but otherwise they have the same semantics as on the command line. Options that can be given on the command line multiple times can be passed as lists like variable=['VAR1:value1', 'VAR2:value2']. If such an option is used only once, it can be given also as a single string like variable='VAR:value'.

Additionally listener option allows passing object directly instead of listener name, e.g. run('tests.robot', listener=Listener()).

To capture stdout and/or stderr streams, pass open file objects in as special keyword arguments stdout and stderr, respectively.

Only options related to the actual test execution have an effect. For example, options related to selecting or modifying test cases or suites (e.g. --include, --name, --prerunmodifier) or creating logs and reports are silently ignored. The output XML generated as part of the execution can be configured, though. This includes disabling it with output=None.

**Example:**

```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.

```python
copy (**attributes)
```

Return shallow copy of this object.

**Parameters attributes** – Attributes to be set for the returned copy automatically. For example, test.copy(name='New name').

See also deepcopy(). The difference between these two is the same as with the standard copy.copy and copy.deepcopy functions that these methods also use internally.

New in Robot Framework 3.0.1.
**deepcopy** (*attributes*)

Return deep copy of this object.

**Parameters**

- **attributes** – Attributes to be set for the returned copy automatically. For example, `test.deepcopy(name='New name')`.

See also `copy()`. The difference between these two is the same as with the standard `copy.copy` and `copy.deepcopy` functions that these methods also use internally.

New in Robot Framework 3.0.1.

**doc**

**filter** (*included_suites=None, included_tests=None, included_tags=None, excluded_tags=None*)

Select test cases and remove others from this suite.

Parameters have the same semantics as `--suite`, `--test`, `--include`, and `--exclude` command line options. All of them can be given as a list of strings, or when selecting only one, as a single string.

Child suites that contain no tests after filtering are automatically removed.

Example:

```python
suite.filter(included_tests=['Test 1', '* Example'],
             included_tags='priority-1')
```

**has_tests**

**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** (*preserve_direct_children=False*)

Removes all child suites not containing any tests, recursively.

**rpa**

**set_tags** (*add=None, remove=None, persist=False*)

Add and/or remove specified tags to the tests in this suite.

**Parameters**

- **add** – Tags to add as a list or, if adding only one, as a single string.
- **remove** – Tags to remove as a list or as a single string. Can be given as patterns where `*` and `?` work as wildcards.
• **persist** – Add/remove specified tags also to new tests added to this suite in the future.

  source

  suites
  Child suites as a *TestSuites* object.

  test_count
  Number of the tests in this suite, recursively.

  tests
  Tests as a *TestCases* object.

  visit(*visitor*)
  Visitor interface entry-point.

class robot.running.model.Variable(*name*, *value*, *source=None*, *lineno=None*, *error=None*)
  Bases: object

  report_invalid_syntax(*message*, *level='ERROR'*)

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*, *lineno=None*, *parent=None*)
  Bases: object

  keywords
  tags
  source

class robot.running.model.Import(*type*, *name*, *args=(), *alias=None*, *source=None*, *lineno=None*)
  Bases: object

  ALLOWED_TYPES = ('Library', 'Resource', 'Variables')

directory

  report_invalid_syntax(*message*, *level='ERROR'*)

class robot.running.model.Imports(*source*, *imports=None*)
  Bases: robot.model.itemlist.ItemList

  append(*item*)
  clear()
  count(*item*)
  create(*args, **kwargs*)
  extend(*items*)
  index(*item, *start_and_end*)
  insert(*index, *item*)
  pop(*index*)
remove(item)
reverse()
sort()
visit(visitor)
library(name, args=(), alias=None, lineno=None)
resource(path, lineno=None)
variables(path, args=(), lineno=None)

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=(), alias=None, notify=True)
    set_search_order(new_order)
    start_test()
    end_test()
    start_suite()
    end_suite(suite)
    start_user_keyword()
    end_user_keyword()
    get_library_instance(libname)
    get_library_instances()
    reload_library(libname_or_instance)
    get_runner(name)

class robot.running.namespace.KeywordStore(resource)
    Bases: object
    get_library(name_or_instance)
    get_runner(name)

class robot.running.namespace.KeywordRecommendationFinder(user_keywords, libraries, resources)
    Bases: object
    recommend_similar_keywords(name)
    static format_recommendations(message, recommendations)
robot.running.outputcapture module

class robot.running.outputcapture.OutputCapturer (library_import=False)
    Bases: object

class robot.running.outputcapture.PythonCapturer (stdout=True)
    Bases: object
        release()

class robot.running.outputcapture.JavaCapturer (stdout=True)
    Bases: object
        release()

robot.running.randomizer module

class robot.running.randomizer.Randomizer (randomize_suites=True, randomize_tests=True, seed=None)
    Bases: robot.model.visitor.SuiteVisitor

    start_suite (suite)
        Called when suite starts. Default implementation does nothing.
        Can return explicit False to stop visiting.

    visit_test (test)
        Implements traversing through the test and its keywords.
        Can be overridden to allow modifying the passed in test without calling start_test() or end_test() nor visiting keywords.

    visit_keyword (kw)
        Implements traversing through the keyword and its child keywords.
        Can be overridden to allow modifying the passed in kw without calling start_keyword() or end_keyword() nor visiting child keywords.

    end_keyword (keyword)
        Called when keyword ends. Default implementation does nothing.

    end_message (msg)
        Called when message ends. Default implementation does nothing.

    end_suite (suite)
        Called when suite ends. Default implementation does nothing.

    end_test (test)
        Called when test ends. Default implementation does nothing.

    start_keyword (keyword)
        Called when keyword starts. Default implementation does nothing.
        Can return explicit False to stop visiting.

    start_message (msg)
        Called when message starts. Default implementation does nothing.
        Can return explicit False to stop visiting.

    start_test (test)
        Called when test starts. Default implementation does nothing.
Can return explicit False to stop visiting.

**visit_message** *(msg)*  
Implements visiting the message.

Can be overridden to allow modifying the passed in *msg* without calling *start_message()* or *end_message()*.

**visit_suite** *(suite)*  
Implements traversing through the suite and its direct children.

Can be overridden to allow modifying the passed in *suite* without calling *start_suite()* or *end_suite()* nor visiting child suites, tests or keywords (setup and teardown) at all.

---

**robot.running.runkwregister module**

**robot.running.runner module**

```python
class robot.running.runner.Runner(output, settings)  
Bases: robot.model.visitor.SuiteVisitor

**start_suite** *(suite)*  
Called when suite starts. Default implementation does nothing.

Can return explicit False to stop visiting.

**end_suite** *(suite)*  
Called when suite ends. Default implementation does nothing.

**visit_test** *(test)*  
Implements traversing through the test and its keywords.

Can be overridden to allow modifying the passed in *test* without calling *start_test()* or *end_test()* nor visiting keywords.

**end_keyword** *(keyword)*  
Called when keyword ends. Default implementation does nothing.

**end_message** *(msg)*  
Called when message ends. Default implementation does nothing.

**end_test** *(test)*  
Called when test ends. Default implementation does nothing.

**start_keyword** *(keyword)*  
Called when keyword starts. Default implementation does nothing.

Can return explicit False to stop visiting.

**start_message** *(msg)*  
Called when message starts. Default implementation does nothing.

Can return explicit False to stop visiting.

**start_test** *(test)*  
Called when test starts. Default implementation does nothing.

Can return explicit False to stop visiting.

**visit_keyword** *(kw)*  
Implements traversing through the keyword and its child keywords.
```
Can be overridden to allow modifying the passed in \texttt{kw} without calling \texttt{start\_keyword()} or \texttt{end\_keyword()} nor visiting child keywords.

\texttt{visit\_message}(\texttt{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}(\texttt{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.

class \texttt{robot.running.runner.ModelCombiner}(\texttt{data, result, **priority})

\texttt{robot.running.signalhandler module}

\texttt{robot.running.status module}

class \texttt{robot.running.status.Failure}

\texttt{robot.running.status.Exit}(\texttt{failure\_mode=False, error\_mode=False, skip\_teardown\_mode=False})

\texttt{failure\_occurred}(\texttt{failure=None, critical=False})

\texttt{error\_occurred}()

\texttt{teardown\_allowed}

class \texttt{robot.running.status.SuiteStatus}(\texttt{parent=None, exit\_on\_failure\_mode=False, exit\_on\_error\_mode=False, skip\_teardown\_on\_exit\_mode=False})

\texttt{critical\_failure\_occurred}()

\texttt{error\_occurred}()

\texttt{failures}

\texttt{message}

\texttt{setup\_executed}(\texttt{failure=None})

\texttt{status}

\texttt{teardown\_allowed}

\texttt{teardown\_executed}(\texttt{failure=None})

class \texttt{robot.running.status.TestStatus}(\texttt{parent, critical})

\texttt{test\_failed}(\texttt{failure})

\texttt{critical\_failure\_occurred}()

\texttt{error\_occurred}()
failures

message

setup_executed(failure=None)

status

tear down allowed

tear down executed(failure=None)

class robot.running.status.TestMessage(status)
Bases: robot.running.status._Message

setup_message = 'Setup failed:
%s'
tear down_message = 'Teardown failed:
%s'
also tear down_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'
tear down_message = 'Suite teardown failed:
%s'
also tear down_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'
tear down_message = 'Parent suite teardown failed:
%s'
also tear down_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.ForRunner(context, templated=False, flavor='IN')`

```python
class robot.running.steprunner.ForRunner(context, templated=False):
    run(data, name=None)
```

```python
class robot.running.steprunner.ForInRunner(context, templated=False):
    Bases: object
    run(data, name=None)
```

```python
class robot.running.steprunner.ForInRangeRunner(context, templated=False):
    Bases: robot.running.steprunner.ForInRunner
    run(data, name=None)
```

```python
class robot.running.steprunner.ForInZipRunner(context, templated=False):
    Bases: robot.running.steprunner.ForInRunner
    run(data, name=None)
```

```python
class robot.running.steprunner.ForInEnumerateRunner(context, templated=False):
    Bases: robot.running.steprunner.ForInRunner
    run(data, name=None)
```

```python
class robot.running.steprunner.InvalidForRunner(context, flavor):
    Bases: robot.running.steprunner.ForInRunner
    Used to send an error from `ForRunner()` if it sees an unexpected error.
    We can’t simply throw a DataError from `ForRunner()` because that happens outside the “with StatusReporter(...)” blocks.
    run(data, name=None)
```

**robot.running.testlibraries module**

```python
robot.running.testlibraries.TestLibrary(name, args=None, variables=None, create_handlers=True)
```

**robot.running.usererrorhandler module**

```python
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 (resource, source_type='Resource file')
    Bases: object
    TEST_CASE_FILE_TYPE = 'Test case file'
    RESOURCE_FILE_TYPE = 'Resource file'
class robot.running.userkeyword.UserKeywordHandler (keyword, libname)
    Bases: object
    longname
    shortdoc
    create_runner (name)
class robot.running.userkeyword.EmbeddedArgumentsHandler (keyword, libname, embedded)
    Bases: robot.running.userkeyword.UserKeywordHandler
    matches (name)
    create_runner (name)
    longname
    shortdoc

robot.running.userkeywordrunner module

class robot.running.userkeywordrunner.UserKeywordRunner (handler, name=None)
    Bases: object
    longname
    libname
    arguments
    run (kw, context)
    dry_run (kw, context)
class robot.running.userkeywordrunner.EmbeddedArgumentsRunner (handler, name)
    Bases: robot.running.userkeywordrunner.UserKeywordRunner
    arguments
    dry_run (kw, context)
    libname
    longname
    run (kw, context)
robot.tidypkg package

Submodules

robot.tidypkg.transformers module

class robot.tidypkg.transformers.Cleaner
    Bases: robot.parsing.model.visitor.ModelTransformer
    Clean up and normalize data.
    Following transformations are made: 1) section headers are normalized to format *** Section Name *** 2)
    setting names are normalize in setting table and in test cases and
    user keywords to format Setting Name or [Setting Name]
    3) settings without values are removed
    4) Empty lines after section headers and within items are removed
    5) For loop declaration and end tokens are normalized to FOR and END
    6) Old style for loop indent (i.e. a cell with only a ‘’ are removed

    visit_CommentSection (section)
    visit_Section (section)
    visit_Statement (statement)
    visit_ForLoop (loop)
    generic_visit (node)
    Called if no explicit visitor function exists for a node.

    visit (node)
    Visit a node.

class robot.tidypkg.transformers.NewlineNormalizer (newline, short_test_name_length)
    Bases: robot.parsing.model.visitor.ModelTransformer
    Normalize new lines in test data
    After this transformation, there is exactly one empty line between each section and between each test or user
    keyword.

    visit_File (node)
    visit_CommentSection (node)
    visit_Section (node)
    visit_TestCaseSection (node)
    visit_TestCase (node)
    visit_KeywordSection (node)
    visit_Keyword (node)
    visit_Statement (statement)
    generic_visit (node)
    Called if no explicit visitor function exists for a node.
visit (node)
  Visit a node.

class robot.tidypkg.transformers.SeparatorNormalizer (use_pipes, space_count)
  Bases: robot.parsing.model.visitor.ModelTransformer
  Make separators and indentation consistent.
  visit_TestCaseSection (section)
  visit_TestCase (node)
  visit_Keyword (node)
  visit_ForLoop (node)
  visit_Statement (statement)
  generic_visit (node)
    Called if no explicit visitor function exists for a node.
  visit (node)
    Visit a node.

class robot.tidypkg.transformers.ColumnAligner (short_test_name_length, widths)
  Bases: robot.parsing.model.visitor.ModelTransformer
    visit_TestCase (node)
    visit_ForLoop (statement)
    visit_Statement (statement)
    align_header (statement)
    align_statement (statement)
    widths_for_line (line)
    should_write_content_after_name (line_pos)
    generic_visit (node)
    Called if no explicit visitor function exists for a node.
  visit (node)
    Visit a node.

class robot.tidypkg.transformers.ColumnWidthCounter
  Bases: robot.parsing.model.visitor.ModelTransformer
    visit_Statement (statement)
    generic_visit (node)
      Called if no explicit visitor function exists for a node.
  visit (node)
      Visit a node.

class robot.tidypkg.transformers.Aligner (short_test_name_length, setting_and_variable_name_length, pipes_mode)
  Bases: robot.parsing.model.visitor.ModelTransformer
  visit_TestCaseSection (section)
  visit_KeywordSection (section)
  visit_Statement (statement)
**generic_visit** *(node)*

Called if no explicit visitor function exists for a node.

**visit** *(node)*

Visit a node.

---

### 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:

```python
from robot import utils
assert utils.Matcher('H?llo').match('Hillo')
```

or:

```python
from robot.utils import Matcher
assert Matcher('H?llo').match('Hillo')
```

**robot.utils.read_rest_data** *(rstfile)*

---

### 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

- *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_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:

--argumentfile can be used to automatically read arguments from a specified file. When --argumentfile is used, the parser always allows using it multiple times. Adding ‘*’ to denote that is thus recommend. A special value ‘stdin’ can be used to read arguments from stdin instead of a file.

--pythonpath can be used to add extra path(s) to sys.path.

--help and --version automatically generate help and version messages. Version is generated based on the tool name and version – see __init__ for information how to set them. Help contains the whole usage given to __init__. Possible <VERSION> text in the usage is replaced with the given version. 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/](http://pyunit.sourceforge.net/). This module can be used freely in same terms as unittest.

**unittest license:**

```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:
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, formatter=None)
 Fail if given objects are unequal as determined by the ‘==’ operator.

robot.utils.asserts.assert_not_equal (first, second, msg=None, values=True, formatter=None)
 Fail if given objects are equal as determined by the ‘==’ operator.
robot.utils.asserts.assert_almost_equal(first, second, places=7, msg=None, values=True)
Fail if the two objects are unequal after rounded to given places.

inequality is determined by object’s difference rounded to the given number of decimal places (default 7) and
comparing to zero. Note that decimal places (from zero) are usually not the same as significant digits (measured
from the most significant digit).

robot.utils.asserts.assert_not_almost_equal(first, second, places=7, msg=None, values=True)
Fail if the two objects are unequal after rounded to given places.

Equality is determined by object’s difference rounded to to the given number of decimal places (default 7) and
comparing to zero. Note that decimal places (from zero) are usually not the same as significant digits (measured
from the most significant digit).

robot.utils.charwidth module

A module to handle different character widths on the console.

Some East Asian characters have width of two on console, and combining characters themselves take no extra space.

See issue 604 [1] for more details about East Asian characters. The issue also contains generate_wild_chars.py script
that was originally used to create _EAST_ASIAN_WILD_CHARS mapping. An updated version of the script is attached
to issue 1096. Big thanks for xieyanbo for the script and the original patch.

Note that Python’s unicodedata module is not used here because importing it takes several seconds on Jython.

robotframework/issues/1096

robot.utils.charwidth.get_char_width(char)

robot.utils.compat module

robot.utils.compat.py2to3(cls)

robot.utils.compat.with_metaclass(meta, *bases)
Create a base class with a metaclass.

robot.utils.compat.isatty(stream)

robot.utils.compress module

robot.utils.compress.compress_text(text)

robot.utils.connectioncache module

class robot.utils.connectioncache.ConnectionCache(no_current_msg='No open connection. ’)
Bases: object

Cache for test libs to use with concurrent connections, processes, etc.

The cache stores the registered connections (or other objects) and allows switching between them using generated
indices or user given aliases. This is useful with any test library where there’s need for multiple concurrent
connections, processes, etc.
This class can, and is, used also outside the core framework by SSHLibrary, Selenium(2)Library, etc. Backwards compatibility is thus important when doing changes.

```python
current = None
    Current active connection.
```

```python
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.
```

```python
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.
```

```python
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.
```

```python
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.
```

```python
empty_cache()
    Empties the connection cache.
    Indexes of the new connections starts from 1 after this.
```

```python
resolve_alias_or_index (alias_or_index)
```

```python
class robot.utils.connectioncache.NoConnection (message)
    Bases: object
```

```python
raise_error()
```

---

**robot.utils.dotdict module**

```python
class robot.utils.dotdict.DotDict (*args, **kwds)
    Bases: collections.OrderedDict
```

```python
clear () → None. Remove all items from od.
```

```python
copy () → a shallow copy of od
```
classmethod fromkeys \((S, v)\) \rightarrow\) New ordered dictionary with keys from S.
   If not specified, the value defaults to None.

get \((k, d)\) \rightarrow D[k] if k in D, else d. d defaults to None.

has_key \((k)\) \rightarrow\) True if D has a key k, else False

items () \rightarrow\) list of (key, value) pairs in od

iteritems ()
   od.iteritems -> an iterator over the (key, value) pairs in od

iterkeys () \rightarrow\) an iterator over the keys in od

itervalues ()
   od.itervalues -> an iterator over the values in od

keys () \rightarrow\) list of keys in od

pop \((k, d)\) \rightarrow 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), return and remove a (key, value) pair.
   Pairs are returned in LIFO order if last is true or FIFO order if false.

setdefault \((k, d)\) \rightarrow od.get(k,d), also set od[k]=d if k not in od

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 values in od

viewitems () \rightarrow a set-like object providing a view on od’s items

viewkeys () \rightarrow a set-like object providing a view on od’s keys

viewvalues () \rightarrow 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)

**class** robot.utils.escaping.Unescaper

Bases: object

unescape(item)

robot.utils.escaping.split_from_equals(string)

**robot.utils.etreewrapper module**

**class** robot.utils.etreewrapper.ETSource(source)

Bases: object
robot.utils.filereader module

```python
class robot.utils.filereader.FileReader(source, accept_text=False)
    Bases: object

    Utility to ease reading different kind of files.

    Supports different sources where to read the data:

    • The source can be a path to a file, either as a string or as a pathlib.Path instance in Python 3. The file itself must be UTF-8 encoded.

    • Alternatively the source can be an already opened file object, including a StringIO or BytesIO object. The file can contain either Unicode text or UTF-8 encoded bytes.

    • The third options is giving the source as Unicode text directly. This requires setting accept_text=True when creating the reader.

In all cases bytes are automatically decoded to Unicode and possible BOM removed.

    read()

    readlines()
```

robot.utils.frange module

```python
robot.utils.frange.frange(*args)
    Like range() but accepts float arguments.
```

robot.utils.htmlformatters module

```python
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)
```
```python
end()

format(lines)

handles(line)

class robot.utils.htmlformatters.HeaderFormatter
Bases: robot.utils.htmlformatters._SingleLineFormatter

match()
    match(string[, pos[, endpos]]) -> match object or None. Matches zero or more characters at the beginning of the string

format_line(line)

add(line)

end()

format(lines)

handles(line)

class robot.utils.htmlformatters.ParagraphFormatter(other_formatters)
Bases: robot.utils.htmlformatters._Formatter

format(lines)

add(line)

end()

handles(line)

class robot.utils.htmlformatters.TableFormatter
Bases: robot.utils.htmlformatters._Formatter

format(lines)

add(line)

end()

handles(line)

class robot.utils.htmlformatters.PreformattedFormatter
Bases: robot.utils.htmlformatters._Formatter

format(lines)

add(line)

end()

handles(line)

class robot.utils.htmlformatters.ListFormatter
Bases: robot.utils.htmlformatters._Formatter

format(lines)

add(line)

end()

handles(line)
```

4.1. robot package
robot.utils.importer module

robot.utils.importer.invalidate_import_caches()

class robot.utils.importer.Importer(type=None, logger=None)
    Bases: object
        import_class_or_module (name, instantiate_with_args=None, return_source=False)
        Imports Python class/module or Java class with given name.
        Class can either live in a module/package or be standalone Java class. In the former case the name is
        something like ‘MyClass’ and in the latter it could be ‘your.package.YourLibrary’. Python classes always
        live in a module, but if the module name is exactly same as the class name then simple ‘MyLibrary’ will
        import a class.
        Python modules can be imported both using format ‘MyModule’ and ‘mymodule.submodule’.
        name can also be a path to the imported file/directory. In that case importing is done using import_class_or_module_by_path
        method.
        If instantiate_with_args is not None, imported classes are instantiated with the specified arguments automatically.

        import_class_or_module_by_path (path, instantiate_with_args=None)
        Import a Python module or Java class using a file system path.
        When importing a Python file, the path must end with ‘.py’ and the actual file must also exist. When
        importing Java classes, the path must end with ‘.java’ or ‘.class’. The class file must exist in both cases
        and in the former case also the source file must exist.
        If instantiate_with_args is not None, imported classes are instantiated with the specified arguments automatically.

class robot.utils.importer.ByPathImporter (logger)
    Bases: robot.utils.importer._Importer
        handles (path)
        import_ (path)

class robot.utils.importer.NonDottedImporter (logger)
    Bases: robot.utils.importer._Importer
        handles (name)
        import_ (name)

class robot.utils.importer.DottedImporter (logger)
    Bases: robot.utils.importer._Importer
        handles (name)
        import_ (name)

robot.utils.markuputils module

robot.utils.markuputils.html_escape (text, linkify=True)
robot.utils.markuputils.xml_escape (text)
robot.utils.markuputils.html_format (text)
robot.utils.markuputils.attribute_escape (attr)
robot.utils.markupwriters module

class robot.utils.markupwriters.HtmlWriter(output, write_empty=True, usage=None)
    Bases: robot.utils.markupwriters._MarkupWriter

    Parameters
    • output – Either an opened, file like object, or a path to the desired output file. In the latter case, the file is created and clients should use close() method to close it.
    • 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)

d end(name, newline=True)

start(name, attrs=None, newline=True)

class robot.utils.markupwriters.XmlWriter(output, write_empty=True, usage=None)
    Bases: robot.utils.markupwriters._MarkupWriter

    Parameters
    • output – Either an opened, file like object, or a path to the desired output file. In the latter case, the file is created and clients should use close() method to close it.
    • 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)

d end(name, newline=True)

start(name, attrs=None, newline=True)

class robot.utils.markupwriters.NullMarkupWriter(**kwargs)
    Bases: object

    Null implementation of the _MarkupWriter interface.

start(**kwargs)

content(**kwargs)

element(**kwargs)

d end(**kwargs)

close(**kwargs)

robot.utils.match module

robot.utils.match.eq(str1, str2, ignore=(), caseless=True, spaceless=True)
```python
class robot.utils.match.Matcher(pattern, ignore=(), caseless=True, spaceless=True, regexp=False)
    Bases: object
    match (string)
    match_any (strings)

class robot.utils.match.MultiMatcher(patterns=None, ignore=(), caseless=True, spaceless=True, match_if_no_patterns=False, regexp=False)
    Bases: object
    match (string)
    match_any (strings)

robot.utils.misc module

robot.utils.misc.roundup (number, ndigits=0, return_type=None)
    Rounds number to the given number of digits.
    Numbers equally close to a certain precision are always rounded away from zero. By default return value is float when ndigits is positive and int otherwise, but that can be controlled with return_type.
    With the built-in round() rounding equally close numbers as well as the return type depends on the Python version.

robot.utils.misc.printable_name (string, code_style=False)
    Generates and returns printable name from the given string.
    Examples: 'simple' -> 'Simple' 'name with spaces' -> 'Name With Spaces' 'more spaces' -> 'More Spaces' 'Cases AND spaces' -> 'Cases AND Spaces' '' -> ''
    If 'code_style' is True:
    'mixedCAPSCamel' -> 'Mixed CAPS Camel' 'camelCaseName' -> 'Camel Case Name' 'under_score_name' -> 'Under Score Name' 'under_and space' -> 'Under And Space' 'miXed CAPS_nAMe' -> 'MiXed CAPS NAMe' '' -> ''

robot.utils.misc.plural_or_not (item)

robot.utils.misc.seq2str (sequence, quote='"', sep=',', lastsep=' and ')
    Returns sequence in format 'item 1', 'item 2' and 'item 3'.

robot.utils.misc.seq2str2 (sequence)
    Returns sequence in format [ item 1 | item 2 | . . . ].

robot.utils.normalizing module

robot.utils.normalizing.normalize (string, ignore=(), caseless=True, spaceless=True)
    Normalizes given string according to given spec.
    By default string is turned to lower case and all whitespace is removed. Additional characters can be removed by giving them in ignore list.

robot.utils.normalizing.normalize_whitespace (string)

robot.utils.normalizing.lower (string)
```
class robot.utils.normalizing.NormalizedDict(initial=None, ignore=(), caseless=True, spaceless=True)

Bases: _abcoll.MutableMapping

Custom dictionary implementation automatically normalizing keys.

Initialized with possible initial value and normalizing spec.

Initial values can be either a dictionary or an iterable of name/value pairs. In the latter case items are added in the given order.

Normalizing spec has exact same semantics as with the normalize() function.

copy()
clear() → None. Remove all items from D.
get(k[, d]) → D[k] if k in D, else d. d defaults to None.
items() → list of D’s (key, value) pairs, as 2-tuples
iteritems() → an iterator over the (key, value) items of D
iterkeys() → an iterator over the keys of D
itervalues() → an iterator over the values of D
keys() → list of D’s keys
pop(k[, d]) → v, remove specified key and return the corresponding value.
    If key is not found, d is returned if given, otherwise KeyError is raised.
popitem() → (k, v), remove and return some (key, value) pair as a 2-tuple; but raise KeyError if D is empty.
setdefault(k[, d]) → D.get(k,d), also set D[k]=d if k not in D
update([E], **F) → None. Update D from mapping/iterable E and F.
    If E present and has a .keys() method, does: for k in E: D[k] = E[k]
    If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v
    In either case, this is followed by: for k, v in F.items(): D[k] = v
values() → list of D’s values

robot.utils.platform module

robot.utils.recommendations module

class robot.utils.recommendations.RecommendationFinder(normalizer=None)

Bases: object

find_and_format(name, candidates, message, max_matches=10)
find(name, candidates, max_matches=10)
    Return a list of close matches to name from candidates.

format(message, recommendations=None)
    Add recommendations to the given message.
    The recommendation string looks like:
robot.util.restreader module

class robot.util.restreader.CaptureRobotData(name, arguments, options, content, 
    lineno, content_offset, block_text, state, 
    state_machine)

    Bases: docutils.parsers.rst.directives.body.CodeBlock

    run()

    add_name(node)
    
    Append self.options[‘name’] to node[‘names’] if it exists.
    
    Also normalize the name string and register it as explicit target.

    assert_has_content()
    
    Throw an ERROR-level DirectiveError if the directive doesn’t have contents.

    debug(message)

    directive_error(level, message)
    
    Return a DirectiveError suitable for being thrown as an exception.
    
    Call “raise self.directive_error(level, message)” from within a directive implementation to return one single
    system message at level level, which automatically gets the directive block and the line number added.
    
    Preferably use the debug, info, warning, error, or severe wrapper methods, e.g. self.
    error(message) to generate an ERROR-level directive error.

    error(message)

    final_argument_whitespace = False

    has_content = True

    info(message)

    option_spec = {'class': <function class_option>, 'name': <function unchanged>, 'number-lines': <function unchanged>}

    optional_arguments = 1

    required_arguments = 0

    severe(message)

    warning(message)

class robot.util.restreader.RobotDataStorage(docstring)

    Bases: object

    add_data(rows)

    get_data()

    has_data()

robot.util.restreader.read_rest_data(rstfile)
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, usage=None)
robot.utils.robotio.binary_file_writer(path=None)
robot.utils.robotio.create_destination_directory(path, usage=None)

robot.utils.robotpath module

robot.utils.robotpath.path_to_url(path)

robot.utils.robotpath.normpath(path, case_normalize=False)
Replacement for os.path.normpath with some enhancements.
1. Convert non-Unicode paths to Unicode using the file system encoding.
2. NFC normalize Unicode paths (affects mainly OSX).
3. Optionally lower-case paths on case-insensitive file systems. That includes Windows and also OSX in default configuration.
4. Turn c: into c:\ on Windows instead of keeping it as c:.

robot.utils.robotpath.abspath(path, case_normalize=False)
Replacement for os.path.abspath with some enhancements and bug fixes.
1. Non-Unicode paths are converted to Unicode using file system encoding.
2. Optionally lower-case paths on case-insensitive file systems. That includes Windows and also OSX in default configuration.
3. Turn c: into c:\ on Windows instead of c:\current\path.

robot.utils.robotpath.get_link_path(target, base)
Returns a relative path to target from base.
If base is an existing file, then its parent directory is considered to be the base. Otherwise base is assumed to be a directory.
The returned path is URL encoded. On Windows returns an absolute path with file: prefix if the target is on a different drive.

robot.utils.robotpath.find_file(path, basedir='.', file_type=None)

4.1. robot package 325
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=" ", timesep=":", millissep=None)
    Returns a timestamp formatted from given time using separators.
    Time can be given either as a timetuple or seconds after epoch.
    Timetuple is (year, month, day, hour, min, sec[, millis]), where parts must be integers and millis is required only when millissep is not None. Notice that this is not 100% compatible with standard Python timetuples which do not have millis.
    Seconds after epoch can be either an integer or a float.

robot.utils.robottime.get_time(format='timestamp', time_=None)
    Return the given or current time in requested format.
    If time is not given, current time is used. How time is returned is determined based on the given ‘format’ string as follows. Note that all checks are case insensitive.
    • If ‘format’ contains word ‘epoch’ the time is returned in seconds after the unix epoch.
    • If ‘format’ contains any of the words ‘year’, ‘month’, ‘day’, ‘hour’, ‘min’ or ‘sec’ only selected parts are returned. The order of the returned parts is always the one in previous sentence and order of words in ‘format’ is not significant. Parts are returned as zero padded strings (e.g. May -> ‘05’).
    • Otherwise (and by default) the time is returned as a timestamp string in format ‘2006-02-24 15:08:31’

robot.utils.robottime.parse_time(timestr)
    Parses the time string and returns its value as seconds since epoch.
    Time can be given in five different formats:
    1) Numbers are interpreted as time since epoch directly. It is possible to use also ints and floats, not only strings containing numbers.
    2) Valid timestamp (‘YYYY-MM-DD hh:mm:ss’ and ‘YYYYMMDD hhmmss’).
    3) ‘NOW’ (case-insensitive) is the current local time.
    4) ‘UTC’ (case-insensitive) is the current time in UTC.
    5) Format ‘NOW - 1 day’ or ‘UTC + 1 hour 30 min’ is the current local/UTC time plus/minus the time specified with the time string.
    Seconds are rounded down to avoid getting times in the future.

robot.utils.robottime.get_timestamp(daysep=",", daytimesep=" ", timesep=":", millissep=".")
robot.utils.robottime.timestamp_to_secs(timestamp, seps=None)
robot.utils.robottime.secs_to_timestamp(secs, seps=None, millis=False)
robot.utils.robottime.get_elapsed_time(start_time, end_time)
    Returns the time between given timestamps in milliseconds.
robot.utils.robottime.elapsed_time_to_string(elapsed, include_millis=True)
    Converts elapsed time in milliseconds to format ‘hh:mm:ss.mil’.
        If include_millis is True, ‘.mil’ part is omitted.
class robot.utils.robottime.TimestampCache
    Bases: object
    get_timestamp(daysep=’,’, daytimesep=’,’, timesep=’:’, millissep=’.’)

robot.utils.robottypes module

robot.utils.robottypes.isTruthy(item)
    Returns True or False depending is the item considered true or not.
    Validation rules:
        • If the value is a string, it is considered false if it is ‘FALSE’, ‘NO’, ‘OFF’, ‘0’, ‘NONE’ or ‘’, case-insensitively. Considering ‘NONE’ false is new in RF 3.0.3 and considering ‘OFF’ and ‘0’ false is new in RF 3.1.
        • Other strings are considered true.
        • Other values are handled by using the standard bool() function.
    Designed to be used also by external test libraries that want to handle Boolean values similarly as Robot Framework itself. See also isFalsy().
robot.utils.robottypes.isFalsy(item)
    Opposite of isTruthy().

robot.utils.robottypes2 module

robot.utils.robottypes2.isInteger(item)
robot.utils.robottypes2.isNumber(item)
robot.utils.robottypes2.isBytes(item)
robot.utils.robottypes2.isString(item)
robot.utils.robottypes2.isUnicode(item)
robot.utils.robottypes2.isPathlike(item)
robot.utils.robottypes2.isListLike(item)
robot.utils.robottypes2.isDictLike(item)
robot.utils.robottypes2.typeName(item)

robot.utils.robottypes3 module

4.1. robot package
robot.utils.setter module

```python
class robot.utils.setter.setter(method)
    Bases: object

class robot.utils.setter.SetterAwareType
    Bases: type

    mro() \rightarrow list
    return a type’s method resolution order
```

robot.utils.sortable module

```python
class robot.utils.sortable.Sortable
    Bases: object

    Base class for sorting based self._sort_key
```

robot.utils.text module

```python
robot.utils.text.cut_long_message(msg)
robot.utils.text.format_assign_message(variable, value, cut_long=True)
robot.utils.text.get_console_length(text)
robot.utils.text.pad_console_length(text, width)
robot.utils.text.split_args_from_name_or_path(name)
robot.utils.text.split_tags_from_doc(doc)
robot.utils.text.getdoc(item)
robot.utils.text.getshortdoc(doc_or_item, linesep='\n')
robot.utils.text.rstrip(string)
```

robot.utils.unic module

```python
robot.utils.unic.unic(item)
robot.utils.unic.prepr(item, width=80)

class robot.utils.unic.PrettyRepr(indent=1, width=80, depth=None, stream=None)
    Bases: pprint.PrettyPrinter

    Handle pretty printing operations onto a stream using a set of configured parameters.

    indent   Number of spaces to indent for each level of nesting.
    width    Attempted maximum number of columns in the output.
    depth    The maximum depth to print out nested structures.
    stream   The desired output stream. If omitted (or false), the standard output stream available at construction will be used.

    format   (object, context, maxlevels, level)
    isreadable   (object)
```
isrecursive(object)
pformat(object)
pprint(object)

robot.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.evaluation module

robot.variables.evaluation.evaluate_expression(expression, variable_store, modules=None, namespace=None)
```python
class robot.variables.evaluation.EvaluationNamespace(variable_store, modules=None, namespace=None):
    Bases: _abcoll.MutableMapping

clear() \rightarrow None. Remove all items from D.

get(k[, d]) \rightarrow D[k] if k in D, else d. d defaults to None.

items() \rightarrow list of D's (key, value) pairs, as 2-tuples

iteritems() \rightarrow an iterator over the (key, value) items of D

iterkeys() \rightarrow an iterator over the keys of D

itervalues() \rightarrow an iterator over the values of D

keys() \rightarrow list of D's keys

pop(k[, d]) \rightarrow 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
```

```python
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)
```

```python
class robot.variables.finders.VariableFinder(variable_store):
    Bases: object

find(variable)

class robot.variables.finders.StoredFinder(store):
    Bases: object
```
 identifiers = '$@&'
    find(name)

class robot.variables.finders.NumberFinder
    Bases: object
    identifiers = '$'
    find(name)

class robot.variables.finders.EmptyFinder
    Bases: object
    identifiers = '$@&'
   classmethod find(key)

class robot.variables.finders.InlinePythonFinder(variables)
    Bases: object
    identifiers = '$@&'
    find(name)

class robot.variables.finders.ExtendedFinder(finder)
    Bases: object
    identifiers = '$@&'
    find(name)

class robot.variables.finders.EnvironmentFinder
    Bases: object
    identifiers = '%'
    find(name)

robot.variables.isvar module

robot.variables.isvar.is_var(string, identifiers='@&', allow_assign_mark=False)
robot.variables.isvar.is_scalar_var(string, allow_assign_mark=False)
robot.variables.isvar.is_list_var(string, allow_assign_mark=False)
robot.variables.isvar.is_dict_var(string, allow_assign_mark=False)
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, message=None, deco_braces=True)
    Raise DataError for missing variable name.
    Return recommendations for similar variable names if any are found.

4.1. robot package
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 variable, its value is returned. Otherwise possible
    variables are replaced with ‘replace_string’. Result may be any object.

    replace_string (item, custom_unescaper=None, ignore_errors=False)
    Replaces variables from a string. Result is always a string.

    Input can also be an already found VariableMatch.

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, custom_unescaper=None, 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)
set_local_variable (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, custom_unescaper=None, 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 ()

del_test ()

start_test ()

del_test ()

start_keyword ()

del_keyword ()

set_global (name, value)

set_suite (name, value)

set_test (name, value)

set_keyword (name, value)

update (variables)

robot.variables.search module

robot.variables.search.search_variable (string, identifiers='$&%*', ignore_errors=False)

class robot.variables.search.VariableMatch (string, identifier=None, base=None, items=(),
start=-1, end=-1)

Bases: object

resolve_base (variables, ignore_errors=False)

name

before

match
after
is_variable
is_list_variable
is_dict_variable
class robot.variables.search.VariableSearcher(identifiers, ignore_errors=False)
   Bases: object
search(string)
variable_state(char)
waiting_item_state(char)
item_state(char)
robot.variables.search.unescape_variable_syntax(item)
class robot.variables.search.VariableIterator(string, identifiers='$@&%*')
   Bases: object

robot.variables.store module
class robot.variables.store.VariableStore(variables)
   Bases: object
   resolve_delayed(item=None)
   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)
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, custom_unescaper=None, 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)

4.1.2 Submodules

4.1.3 robot.errors module

Exceptions and return codes used internally.
External libraries should not used exceptions defined here.

exception robot.errors.RobotError (message=", details=")
Bases: exceptions.Exception

Base class for Robot Framework errors.
Do not raise this method but use more specific errors instead.

message
args

exception robot.errors.FrameworkError (message='', details='')
    Bases: robot.errors.RobotError

Can be used when the core framework goes to unexpected state.

It is good to explicitly raise a FrameworkError if some framework component is used incorrectly. This is pretty much same as ‘Internal Error’ and should of course never happen.

args

message

exception robot.errors.DataError (message='', details='')
    Bases: robot.errors.RobotError

Used when the provided test data is invalid.

DataErrors are not caught by keywords that run other keywords (e.g. Run Keyword And Expect Error).

args

message

exception robot.errors.VariableError (message='', details='')
    Bases: robot.errors.DataError

Used when variable does not exist.

VariableErrors are caught by keywords that run other keywords (e.g. Run Keyword And Expect Error).

args

message

exception robot.errors.KeywordError (message='', details='')
    Bases: robot.errors.DataError

Used when no keyword is found or there is more than one match.

KeywordErrors are caught by keywords that run other keywords (e.g. Run Keyword And Expect Error).

args

message

exception robot.errors.TimeoutError (message='', test_timeout=True)
    Bases: robot.errors.RobotError

Used when a test or keyword timeout occurs.

This exception is handled specially so that execution of the current test is always stopped immediately and it is not caught by keywords executing other keywords (e.g. Run Keyword And Expect Error).

keyword_timeout

args

message

exception robot.errors.Information (message='', details='')
    Bases: robot.errors.RobotError

Used by argument parser with –help or –version.

args

message
exception robot.errors.ExecutionStatus

Bases: robot.errors.RobotError

Base class for exceptions communicating status in test execution.

timeout
dont_continue
continue_on_failure
can_continue (teardown=False, templated=False, dry_run=False)
get_errors()
status
args
message

exception robot.errors.ExecutionFailed

Bases: robot.errors.ExecutionStatus

Used for communicating failures in test execution.

args
can_continue (teardown=False, templated=False, dry_run=False)
continue_on_failure
dont_continue
get_errors()
message
status
timeout

exception robot.errors.HandlerExecutionFailed

Bases: robot.errors.ExecutionFailed

args
can_continue (teardown=False, templated=False, dry_run=False)
continue_on_failure
dont_continue
get_errors()
message
status
timeout

exception robot.errors.ExecutionFailures

Bases: robot.errors.ExecutionFailed

get_errors()
Robot Framework Documentation, Release 3.2b3.dev1

```python
args
can_continue (teardown=False, templated=False, dry_run=False)
continue_on_failure
dont_continue
message
status
timeout

exception robot.errors.UserKeywordExecutionFailed (run_errors=None, teardown_errors=None)
Bases: robot.errors.ExecutionFailures

args
can_continue (teardown=False, templated=False, dry_run=False)
continue_on_failure
dont_continue
get_errors ()
message
status
timeout

exception robot.errors.ExecutionPassed (message=None, **kwargs)
Bases: robot.errors.ExecutionStatus

Base class for all exceptions communicating that execution passed.
Should not be raised directly, but more detailed exceptions used instead.

set_earlier_failures (failures)
earlier_failures
status
args
can_continue (teardown=False, templated=False, dry_run=False)
continue_on_failure
dont_continue
get_errors ()
message
timeout

exception robot.errors.PassExecution (message)
Bases: robot.errors.ExecutionPassed

Used by ‘Pass Execution’ keyword.

args
can_continue (teardown=False, templated=False, dry_run=False)
continue_on_failure
```
```
dont_continue
earlier_failures
get_errors()
message
set_earlier_failures(failures)
status
timeout

excetion robot.errors.ContinueForLoop (message=None, **kwargs)
   Bases: robot.errors.ExecutionPassed
   Used by ‘Continue For Loop’ keyword.
   args
can_continue (teardown=False, templated=False, dry_run=False)
continue_on_failure
dont_continue
earlier_failures
get_errors()
message
set_earlier_failures(failures)
status
timeout

excetion robot.errors.ExitForLoop (message=None, **kwargs)
   Bases: robot.errors.ExecutionPassed
   Used by ‘Exit For Loop’ keyword.
   args
can_continue (teardown=False, templated=False, dry_run=False)
continue_on_failure
dont_continue
earlier_failures
get_errors()
message
set_earlier_failures(failures)
status
timeout

excetion robot.errors.ReturnFromKeyword (return_value=None, failures=None)
   Bases: robot.errors.ExecutionPassed
   Used by ‘Return From Keyword’ keyword.
   args
```
can_continue (teardown=False, templated=False, dry_run=False)
continue_on_failure
dont_continue
earlier_failures
get_errors ()
message
set_earlier_failures (failures)
status
timeout

exception robot.errors.RemoteError (message=", details=", fatal=False, continuable=False)
Bases: robot.errors.RobotError
Used by Remote library to report remote errors.
args
message

4.1.4 robot.jarrunner module

4.1.5 robot.libdoc module

Module implementing the command line entry point for the Libdoc tool.
This module can be executed from the command line using the following approaches:

```
python -m robot.libdoc
python path/to/robot/libdoc.py
```

Instead of python it is possible to use also other Python interpreters.
This module also provides libdoc() and libdoc_cli() functions that can be used programmatically. Other code is for internal usage.
Libdoc itself is implemented in the libdocpkg package.

class robot.libdoc.LibDoc
Bases: robot.utils.application.Application
validate (options, arguments)
main (args, name=", version=", format=None, docformat=None)
console (msg)
execute (*arguments, **options)
execute_cli (cli_arguments, exit=True)
parse_arguments (cli_args)

Public interface for parsing command line arguments.

Parameters cli_args – Command line arguments as a list
Returns options (dict), arguments (list)
Raises Information when –help or –version used
Raises DataError when parsing fails

robot.libdoc.libdoc_cli(arguments)
Executes Libdoc similarly as from the command line.

Parameters arguments – Command line arguments as a list of strings.
For programmatic usage the libdoc() function is typically better. It has a better API for that usage and does not call sys.exit() like this function.

Example:
```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 programatically. Other code is for internal usage.

```python
class robot.rebot.Rebot
    Bases: robot.run.RobotFramework

    main(datasources, **options)
    console(msg)
    execute(*arguments, **options)
    execute_cli(cli_arguments, exit=True)
    parse_arguments(cli_args)

    Public interface for parsing command line arguments.
    Parameters
    cli_args – Command line arguments as a list
    Returns  options (dict), arguments (list)
    Raises  Information when --help or --version used
    Raises  DataError when parsing fails

    validate(options, arguments)

robot.rebot.rebot_cli(arguments=None, exit=True)
    Command line execution entry point for post-processing outputs.
    Parameters
    • arguments – Command line options and arguments as a list of strings. Starting from RF 3.1, defaults to sys.argv[1:] if not given.
    • exit – If True, call sys.exit with the return code denoting execution status, otherwise just return the rc. New in RF 3.0.1.

Entry point used when post-processing outputs from the command line, but can also be used by custom scripts. Especially useful if the script itself needs to accept same arguments as accepted by Rebot, because the script can just pass them forward directly along with the possible default values it sets itself.

Example:

```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.

```python
robot.rebot.rebot(*outputs, **options)
    Programmatic entry point for post-processing outputs.
    Parameters
```
• **outputs** – Paths to Robot Framework output files similarly as when running the `rebot` command on the command line.

• **options** – Options to configure processing outputs. Accepted options are mostly same as normal command line options to the `rebot` command. Option names match command line option long names without hyphens so that, for example, `--name` becomes `name`.

The semantics related to passing options are exactly the same as with the `run()` function. See its documentation for more details.

Examples:

```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:

```bash
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:

```bash
python -m robot.run
python path/to/robot/run.py
```

Instead of `python` it is possible to use also other Python interpreters. This module is also used by the installed `robot` start-up script.

This module also provides `run()` and `run_cli()` functions that can be used programmatically. Other code is for internal usage.

```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=None, exit=True)

Command line execution entry point for running tests.

Parameters

- **arguments** – Command line options and arguments as a list of strings. Starting from RF 3.1, defaults to `sys.argv[1:]` if not given.
- **exit** – If True, call `sys.exit` with the return code denoting execution status, otherwise just return the rc. New in RF 3.0.1.

Entry point used when running tests from the command line, but can also be used by custom scripts that execute tests. Especially useful if the script itself needs to accept same arguments as accepted by Robot Framework, because the script can just pass them forward directly along with the possible default values it sets itself.

Example:

```python
from robot import run_cli

# Run tests and return the return code.
rc = run_cli(['--name', 'Example', 'tests.robot'], exit=False)

# Run tests and exit to the system automatically.
run_cli(['--name', 'Example', 'tests.robot'])
```

See also the `run()` function that allows setting options as keyword arguments like name="Example" and generally has a richer API for programmatic test execution.

robot.run.run(*tests, **options)

Programmatic entry point for running tests.

Parameters

- **tests** – Paths to test case files/directories to be executed similarly as when running the `robot` command on the command line.
- **options** – Options to configure and control execution. Accepted options are mostly same as normal command line options to the `robot` command. Option names match command line option long names without hyphens so that, for example, `--name` becomes `name`.

Most options that can be given from the command line work. An exception is that options `--pythonpath`, `--argumentfile`, `--help` and `--version` are not supported.

Options that can be given on the command line multiple times can be passed as lists. For example, `include=['tag1', 'tag2']` is equivalent to `--include tag1 --include tag2`. If such options are used only once, they can be given also as a single string like `include='tag'`.

Options that accept no value can be given as Booleans. For example, `dryrun=True` is same as using the `--dryrun` option.

Options that accept string `NONE` as a special value can also be used with Python `None`. For example, using `log=None` is equivalent to `--log NONE`.

`listener`, `prerunmodifier` and `prerobotmodifier` 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:

```plaintext
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:

```plaintext
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)

robot.testdoc.TestSuiteFactory (datasources, **options)

class robot.testdoc.TestdocModelWriter (output, suite, title=None)
    Bases: robot.htmldata.htmlfilewriter.ModelWriter
    write (line)
    write_data ()
    handles (line)
```
class robot.testdoc.JsonConverter(output_path=None)
    Bases: object
    convert(suite)

robot.testdoc.testdoc_cli(arguments)
    Executes Testdoc similarly as from the command line.

    Parameters arguments – command line arguments as a list of strings.

    For programmatic usage the testdoc() function is typically better. It has a better API for that and does not call sys.exit() like this function.

    Example:

    from robot.testdoc import testdoc_cli
    testdoc_cli(['--title', 'Test Plan', 'mytests', 'plan.html'])

robot.testdoc.testdoc(*arguments, **options)
    Executes Testdoc programmatically.

    Arguments and options have same semantics, and options have same names, as arguments and options to Testdoc.

    Example:

    from robot.testdoc import testdoc
    testdoc('mytests', 'plan.html', title='Test Plan')

4.1.10 robot.tidy module

Module implementing the command line entry point for the Tidy tool.

This module can be executed from the command line using the following approaches:

    python -m robot.tidy
    python path/to/robot/tidy.py

Instead of python it is possible to use also other Python interpreters.

This module also provides Tidy class and tidy_cli() function that can be used programmatically. Other code is for internal usage.

class robot.tidy.Tidy(space_count=4, use_pipes=False, line_separator='n')
    Bases: robot.parsing.suitestructure.SuiteStructureVisitor

    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, outpath=None)
        Tidy a file.

        Parameters

        • path – Path of the input file.

        • outpath – 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 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
visit_file(file)
visit_directory(directory)
end_directory(structure)
start_directory(structure)
```

**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, usepipes=False, spacecount=4, lineseparator='\n')
```

**validate**

```python
 opts, args)
```

**console**

```python
 msg)
```

**execute**

```python
 *arguments, **options)
```

**execute_cli**

```python
 cli_arguments, exit=True)
```

**parse_arguments**

```python
 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

**class** `robot.tidy.ArgumentValidator`

**Bases:** `object`

```python
 mode_and_args(args, recursive, inplace, **others)
```

```python
 line_sep(lineseparator, **others)
```

```python
 spacecount(spacecount)
```

**robot.tidy.tidy_cli( arguments)**

Executes `Tidy` similarly as from the command line.

**Parameters**

- `arguments` – Command line arguments as a list of strings.

Example:
from robot.tidy import tidy_cli

tidy_cli(['--spacecount', '2', 'tests.robot'])

### 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
<table>
<thead>
<tr>
<th>Module</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>robot</td>
<td>9</td>
</tr>
<tr>
<td>robot.api</td>
<td>7</td>
</tr>
<tr>
<td>robot.api.deco</td>
<td>12</td>
</tr>
<tr>
<td>robot.api.logger</td>
<td>14</td>
</tr>
<tr>
<td>robot.conf</td>
<td>15</td>
</tr>
<tr>
<td>robot.conf.gatherfalled</td>
<td>16</td>
</tr>
<tr>
<td>robot.conf.settings</td>
<td>18</td>
</tr>
<tr>
<td>robot.errors</td>
<td>335</td>
</tr>
<tr>
<td>robot.htmldata</td>
<td>20</td>
</tr>
<tr>
<td>robot.htmldata.htmlfilewriter</td>
<td>20</td>
</tr>
<tr>
<td>robot.htmldata.jsonwriter</td>
<td>20</td>
</tr>
<tr>
<td>robot.htmldata.normaltemplate</td>
<td>21</td>
</tr>
<tr>
<td>robot.htmldata.template</td>
<td>21</td>
</tr>
<tr>
<td>robot.libdoc</td>
<td>340</td>
</tr>
<tr>
<td>robot.libdocpkg</td>
<td>21</td>
</tr>
<tr>
<td>robot.libdocpkg.builder</td>
<td>22</td>
</tr>
<tr>
<td>robot.libdocpkg.consoleviewer</td>
<td>22</td>
</tr>
<tr>
<td>robot.libdocpkg.htmlwriter</td>
<td>22</td>
</tr>
<tr>
<td>robot.libdocpkg.javabuilder</td>
<td>23</td>
</tr>
<tr>
<td>robot.libdocpkg.model</td>
<td>23</td>
</tr>
<tr>
<td>robot.libdocpkg.output</td>
<td>23</td>
</tr>
<tr>
<td>robot.libdocpkg.robotbuilder</td>
<td>23</td>
</tr>
<tr>
<td>robot.libdocpkg.specbuilder</td>
<td>24</td>
</tr>
<tr>
<td>robot.libdocpkg.writer</td>
<td>24</td>
</tr>
<tr>
<td>robot.libdocpkg.xmlwriter</td>
<td>24</td>
</tr>
<tr>
<td>robot.libdocpkg</td>
<td>24</td>
</tr>
<tr>
<td>robot.libraries</td>
<td>48</td>
</tr>
<tr>
<td>robot.libraries.Collections</td>
<td>48</td>
</tr>
<tr>
<td>robot.libraries.DateTime</td>
<td>54</td>
</tr>
<tr>
<td>robot.libraries.Dialogs</td>
<td>59</td>
</tr>
<tr>
<td>robot.librariesdialogs_py</td>
<td>106</td>
</tr>
<tr>
<td>robot.libraries.Easter</td>
<td>60</td>
</tr>
<tr>
<td>robot.libraries.OperatingSystem</td>
<td>61</td>
</tr>
<tr>
<td>robot.libraries.Process</td>
<td>70</td>
</tr>
<tr>
<td>robot.libraries.Remote</td>
<td>76</td>
</tr>
<tr>
<td>robot.libraries.Reserved</td>
<td>78</td>
</tr>
<tr>
<td>robot.libraries.Screenshot</td>
<td>78</td>
</tr>
<tr>
<td>robot.libraries.String</td>
<td>80</td>
</tr>
<tr>
<td>robot.libraries.Telnet</td>
<td>85</td>
</tr>
<tr>
<td>robot.libraries.XML</td>
<td>95</td>
</tr>
<tr>
<td>robot.model</td>
<td>171</td>
</tr>
<tr>
<td>robot.model.configurer</td>
<td>171</td>
</tr>
<tr>
<td>robot.model.criticality</td>
<td>172</td>
</tr>
<tr>
<td>robot.model.filter</td>
<td>172</td>
</tr>
<tr>
<td>robot.model.itemlist</td>
<td>175</td>
</tr>
<tr>
<td>robot.model.keyword</td>
<td>175</td>
</tr>
<tr>
<td>robot.model.message</td>
<td>177</td>
</tr>
<tr>
<td>robot.model.metadata</td>
<td>179</td>
</tr>
<tr>
<td>robot.model.modelobject</td>
<td>179</td>
</tr>
<tr>
<td>robot.model.modifier</td>
<td>180</td>
</tr>
<tr>
<td>robot.model.namepatterns</td>
<td>181</td>
</tr>
<tr>
<td>robot.model.statistics</td>
<td>181</td>
</tr>
<tr>
<td>robot.model.stats</td>
<td>182</td>
</tr>
<tr>
<td>robot.model.suitestatistics</td>
<td>184</td>
</tr>
<tr>
<td>robot.model.tags</td>
<td>185</td>
</tr>
<tr>
<td>robot.model.tagsetter</td>
<td>185</td>
</tr>
<tr>
<td>robot.model.tagstatistics</td>
<td>186</td>
</tr>
<tr>
<td>robot.model.testcase</td>
<td>187</td>
</tr>
<tr>
<td>robot.model.testsuite</td>
<td>189</td>
</tr>
<tr>
<td>robot.model.totalstatistics</td>
<td>191</td>
</tr>
<tr>
<td>robot.model.visitor</td>
<td>192</td>
</tr>
<tr>
<td>robot.output</td>
<td>194</td>
</tr>
<tr>
<td>robot.output.console</td>
<td>194</td>
</tr>
<tr>
<td>robot.output.console.dotted</td>
<td>195</td>
</tr>
<tr>
<td>robot.output.console.highlighting</td>
<td>196</td>
</tr>
<tr>
<td>robot.output.console.quiet</td>
<td>197</td>
</tr>
<tr>
<td>robot.output.console.verbose</td>
<td>197</td>
</tr>
<tr>
<td>robot.output.debugfile</td>
<td>198</td>
</tr>
<tr>
<td>robot.output.filelogger</td>
<td>198</td>
</tr>
<tr>
<td>robot.output.librarylogger</td>
<td>198</td>
</tr>
<tr>
<td>robot.output.listenerarguments</td>
<td>199</td>
</tr>
<tr>
<td>robot.output.listenermethods</td>
<td>199</td>
</tr>
<tr>
<td>robot.output.listeners</td>
<td>200</td>
</tr>
<tr>
<td>robot.output.logger</td>
<td>200</td>
</tr>
<tr>
<td>robot.output.loggerhelper</td>
<td>202</td>
</tr>
<tr>
<td>robot.output.output</td>
<td>203</td>
</tr>
<tr>
<td>robot.output.pyloggingconf</td>
<td>203</td>
</tr>
<tr>
<td>robot.output.stdoutlogspli</td>
<td>205</td>
</tr>
<tr>
<td>Module</td>
<td>Page</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>robot.output.xmllogger</td>
<td>205</td>
</tr>
<tr>
<td>robot.parsing</td>
<td>206</td>
</tr>
<tr>
<td>robot.parsing.lexer</td>
<td>207</td>
</tr>
<tr>
<td>robot.parsing.lexer.context</td>
<td>207</td>
</tr>
<tr>
<td>robot.parsing.lexer.settings</td>
<td>209</td>
</tr>
<tr>
<td>robot.parsing.lexer.tokens</td>
<td>211</td>
</tr>
<tr>
<td>robot.parsing.model</td>
<td>214</td>
</tr>
<tr>
<td>robot.parsing.model.blocks</td>
<td>214</td>
</tr>
<tr>
<td>robot.parsing.model.statements</td>
<td>217</td>
</tr>
<tr>
<td>robot.parsing.model.visitor</td>
<td>237</td>
</tr>
<tr>
<td>robot.parsing.suitestructure</td>
<td>237</td>
</tr>
<tr>
<td>robot.pythonpathsetter</td>
<td>341</td>
</tr>
<tr>
<td>robot.rebot</td>
<td>341</td>
</tr>
<tr>
<td>robot.reporting</td>
<td>238</td>
</tr>
<tr>
<td>robot.reporting.expandkeywordmatcher</td>
<td>238</td>
</tr>
<tr>
<td>robot.reporting.jsbuildingcontext</td>
<td>238</td>
</tr>
<tr>
<td>robot.reporting.jsexecutionresult</td>
<td>239</td>
</tr>
<tr>
<td>robot.reporting.jsmodelbuilders</td>
<td>239</td>
</tr>
<tr>
<td>robot.reporting.jswriter</td>
<td>240</td>
</tr>
<tr>
<td>robot.reporting.logreportwriters</td>
<td>240</td>
</tr>
<tr>
<td>robot.reporting.outputwriter</td>
<td>240</td>
</tr>
<tr>
<td>robot.reporting.resultwriter</td>
<td>242</td>
</tr>
<tr>
<td>robot.reporting.stringcache</td>
<td>242</td>
</tr>
<tr>
<td>robot.reporting.junitwriter</td>
<td>243</td>
</tr>
<tr>
<td>robot.result</td>
<td>245</td>
</tr>
<tr>
<td>robot.result.configurer</td>
<td>246</td>
</tr>
<tr>
<td>robot.result.executionerrors</td>
<td>247</td>
</tr>
<tr>
<td>robot.result.executionresult</td>
<td>247</td>
</tr>
<tr>
<td>robot.result.flattenkeywordmatcher</td>
<td>250</td>
</tr>
<tr>
<td>robot.result.keywordremover</td>
<td>250</td>
</tr>
<tr>
<td>robot.result.merger</td>
<td>256</td>
</tr>
<tr>
<td>robot.result.messagefilter</td>
<td>257</td>
</tr>
<tr>
<td>robot.result.model</td>
<td>258</td>
</tr>
<tr>
<td>robot.result.resultbuilder</td>
<td>265</td>
</tr>
<tr>
<td>robot.result.suitetestdownfailed</td>
<td>267</td>
</tr>
<tr>
<td>robot.result.visitor</td>
<td>269</td>
</tr>
<tr>
<td>robot.result.xmlelementhandlers</td>
<td>271</td>
</tr>
<tr>
<td>robot.run</td>
<td>343</td>
</tr>
<tr>
<td>robot.running</td>
<td>274</td>
</tr>
<tr>
<td>robot.running.arguments</td>
<td>276</td>
</tr>
<tr>
<td>robot.running.arguments.argumentconverters</td>
<td>276</td>
</tr>
<tr>
<td>robot.running.arguments.argumentmapper</td>
<td>276</td>
</tr>
<tr>
<td>robot.running.arguments.argumentparser</td>
<td>276</td>
</tr>
<tr>
<td>robot.running.arguments.argumentresolver</td>
<td>277</td>
</tr>
<tr>
<td>robot.running.arguments.argumentspec</td>
<td>277</td>
</tr>
<tr>
<td>robot.running.arguments.argumentvalidator</td>
<td>278</td>
</tr>
<tr>
<td>robot.running.arguments.embedded</td>
<td>278</td>
</tr>
<tr>
<td>robot.running.arguments.typeconverters</td>
<td>278</td>
</tr>
<tr>
<td>robot.running.arguments.typevalidator</td>
<td>284</td>
</tr>
<tr>
<td>robot.running.builder</td>
<td>284</td>
</tr>
<tr>
<td>robot.running.builder.builders</td>
<td>284</td>
</tr>
<tr>
<td>robot.running.builder.parsers</td>
<td>285</td>
</tr>
<tr>
<td>robot.running.builder.testsettings</td>
<td>286</td>
</tr>
<tr>
<td>robot.running.builder.transformers</td>
<td>286</td>
</tr>
<tr>
<td>robot.running.context</td>
<td>289</td>
</tr>
<tr>
<td>robot.running.dynamicmethods</td>
<td>290</td>
</tr>
<tr>
<td>robot.running.handlers</td>
<td>290</td>
</tr>
<tr>
<td>robot.running.handlerstore</td>
<td>291</td>
</tr>
<tr>
<td>robot.running.importer</td>
<td>291</td>
</tr>
<tr>
<td>robot.running.librarykeywordrunner</td>
<td>291</td>
</tr>
<tr>
<td>robot.running.libraryscopes</td>
<td>292</td>
</tr>
<tr>
<td>robot.running.model</td>
<td>293</td>
</tr>
<tr>
<td>robot.running.namespace</td>
<td>301</td>
</tr>
<tr>
<td>robot.running.outputcapture</td>
<td>302</td>
</tr>
<tr>
<td>robot.running.randomizer</td>
<td>302</td>
</tr>
<tr>
<td>robot.running.runkwregister</td>
<td>303</td>
</tr>
<tr>
<td>robot.running.runner</td>
<td>303</td>
</tr>
<tr>
<td>robot.running.signalhandler</td>
<td>304</td>
</tr>
<tr>
<td>robot.running.status</td>
<td>304</td>
</tr>
<tr>
<td>robot.running.statuareporter</td>
<td>305</td>
</tr>
<tr>
<td>robot.running.steprunner</td>
<td>305</td>
</tr>
<tr>
<td>robot.running.testlibraries</td>
<td>306</td>
</tr>
<tr>
<td>robot.running.timeouts</td>
<td>288</td>
</tr>
<tr>
<td>robot.running.timeout.posix</td>
<td>289</td>
</tr>
<tr>
<td>robot.running.timeout.windows</td>
<td>289</td>
</tr>
<tr>
<td>robot.running.usererrorhandler</td>
<td>306</td>
</tr>
<tr>
<td>robot.running.userkeyword</td>
<td>307</td>
</tr>
<tr>
<td>robot.running.userkeywordrunner</td>
<td>307</td>
</tr>
<tr>
<td>robot.running.testdoc</td>
<td>345</td>
</tr>
<tr>
<td>robot.tidy</td>
<td>346</td>
</tr>
<tr>
<td>robot.tidypkg</td>
<td>308</td>
</tr>
<tr>
<td>robot.tidypkg.transformers</td>
<td>308</td>
</tr>
<tr>
<td>robot.tidypkg.transformers</td>
<td>308</td>
</tr>
<tr>
<td>robot.tidypkg</td>
<td>308</td>
</tr>
<tr>
<td>robot.tidypkg.transformers</td>
<td>308</td>
</tr>
<tr>
<td>robot.tidypkg</td>
<td>308</td>
</tr>
<tr>
<td>robot.utils</td>
<td>310</td>
</tr>
<tr>
<td>robot.utils.application</td>
<td>310</td>
</tr>
<tr>
<td>robot.utils.argumentparser</td>
<td>311</td>
</tr>
<tr>
<td>robot.utils.asserts</td>
<td>312</td>
</tr>
<tr>
<td>robot.utils.charsize</td>
<td>314</td>
</tr>
<tr>
<td>robot.utils.compat</td>
<td>314</td>
</tr>
<tr>
<td>robot.utils.compress</td>
<td>314</td>
</tr>
<tr>
<td>robot.utils.connectioncache</td>
<td>314</td>
</tr>
<tr>
<td>robot.utils.dotdict</td>
<td>315</td>
</tr>
<tr>
<td>robot.utils.encoding</td>
<td>316</td>
</tr>
<tr>
<td>robot.utils.etreewrapper</td>
<td>317</td>
</tr>
<tr>
<td>robot.utils.filereader</td>
<td>318</td>
</tr>
<tr>
<td>robot.utils.frange</td>
<td>318</td>
</tr>
<tr>
<td>robot.utils.htmlformatters</td>
<td>318</td>
</tr>
</tbody>
</table>
robot.utils.importer, 320
robot.utils.markuputils, 320
robot.utils.markupwriters, 321
robot.utils.match, 321
robot.utils.misc, 322
robot.utils.normalizing, 322
robot.utils.platform, 323
robot.utils.recommendations, 323
robot.utils.restreader, 324
robot.utils.robotenv, 325
robot.utils.robotinspect, 325
robot.utils.robotio, 325
robot.utils.robotpath, 325
robot.utils.robottime, 326
robot.utils.robottypes, 327
robot.utils.robottypes2, 327
robot.utils.setter, 328
robot.utils.sortable, 328
robot.utils.text, 328
robot.utils.unic, 328
robot.variables, 329
robot.variables.assigner, 329
robot.variables.evaluation, 329
robot.variables.filesetter, 330
robot.variables.finders, 330
robot.variables.isvar, 331
robot.variables.notfound, 331
robot.variables.replacer, 332
robot.variables.scopes, 332
robot.variables.search, 333
robot.variables.store, 334
robot.variables.tablesetter, 334
robot.variables.variables, 335
robot.version, 348
Index

A

abc (robot.running.arguments.typeconverters.BooleanConverter
attribute), 278

accept_gzip_encoding (robot.libraries.Remote.TimeoutHTTPSTransport
attribute), 77

accept_gzip_encoding
(robot.libraries.Remote.TimeoutHTTPTransport
attribute), 77

acquire () (robot.output.pyloggingconf.RobotHandler
method), 204

active
(robot.running.timeouts.KeywordTimeout attribute), 289

active
(robot.running.timeouts.TestTimeout attribute), 288

add () (robot.model.tags.Tags method), 185

add () (robot.parsing.model.blocks.Body method), 216

add () (robot.reporting.stringcache.StringCache
method), 243

add () (robot.result.executionerrors.ExecutionErrors
method), 247

add () (robot.running.handlerstore.HandlerStore
method), 291

add () (robot.running.importer.ImportCache method), 291

add () (robot.utils.htmlformatters.HeaderFormatter
method), 319

add () (robot.utils.htmlformatters.ListFormatter
method), 319

add () (robot.utils.htmlformatters.ParagraphFormatter
method), 319

add () (robot.utils.htmlformatters.RulerFormatter
method), 319

add () (robot.utils.htmlformatters.TableFormatter
method), 319

add () (robot.utils.htmlformatters.PreformattedFormatter
method), 319

add_element () (robot.libraries.XML.XML
method), 320

add_data () (robot.utils.restreader.RobotDataStorage
method), 324

add_element () (robot.variables.store.VariableStore
method), 334

abspath() (in module robot.utils.robotpath), 325
add_name() (robot.util.restreader.CaptureRobotData method), 324
add_path() (in module robot.pythonpathsetter), 341
add_result() (robot.result.executionresult.CombinedResult method), 248
add_stat() (robot.model.stats.SuiteStat method), 183
add_tags (robot.model.configurer.SuiteConfigurer attribute), 171
add_tags (robot.result.configurer.SuiteConfigurer attribute), 246
add_test() (robot.model.stats.CombinedTagStat method), 184
add_test() (robot.model.stats.CriticalTagStat method), 184
add_test() (robot.model.stats.Stat method), 182
add_test() (robot.model.stats.TagStat method), 183
add_test() (robot.model.stats.TotalStat method), 183
add_test() (robot.model.suitestatistics.SuiteStatisticsBuilder aliases) (robot.parsing.lexer.settings.TestCaseSettings attribute), 210
add_test() (robot.model.tagstatistics.TagStatisticsBuilder aliases) (robot.parsing.lexer.settings.TestCaseSettings attribute), 210
add_time_to_date() (in module robot.libraries.DateTime), 58
add_time_to_time() (in module robot.libraries.DateTime), 59
addFilter() (robot.output.pyloggingconf.RobotHandler aliases) (robot.running.arguments.typeconverters.DateTimeConverter attribute), 281
after (robot.variables.search.VariableMatch attribute), 333
after() (robot.libraries.dialogs_py.InputDialog method), 119
after() (robot.libraries.dialogs_py.MessageDialog method), 106
after() (robot.libraries.dialogs_py.MultilineInputDialog method), 145
after() (robot.libraries.dialogs_py.PassFailDialog method), 158
after() (robot.libraries.dialogs_py.SelectionDialog method), 132
after_cancel() (robot.libraries.dialogs_py.InputDialog aliases) (robot.running.arguments.typeconverters.IntegerConverter attribute), 279
after_cancel() (robot.libraries.dialogs_py.SelectionDialog aliases) (robot.running.arguments.typeconverters.NoneConverter attribute), 284
after_idle() (robot.libraries.dialogs_py.MessageDialog method), 106
after_idle() (robot.libraries.dialogs_py.MultilineInputDialog method), 145
after_idle() (robot.libraries.dialogs_py.PassFailDialog method), 158
after_idle() (robot.libraries.dialogs_py.SelectionDialog method), 132
after_idle() (robot.libraries.pythonpathsetter.CaptureRobotData method), 119
alias (robot.parsing.model.statements.LibraryImport attribute), 222
aliases (robot.parsing.lexer.settings.KeywordSettings attribute), 211
Index
ArgumentCoercer (class in robot.libraries.Remote), 77
ArgumentConverter (class in robot.running.arguments.argumentconverter), 276
ArgumentHandler (class in robot.result.xmlelementhandlers), 274
ArgumentMapper (class in robot.running.arguments.argumentmapper), 276
ArgumentParser (class in robot.running.arguments.argumentparser), 311
ArgumentResolver (class in robot.running.arguments.argumentresolver), 277
Arguments (class in robot.parsing.model.statements), 232
ARGUMENTS (robot.parsing.lexer.tokens.EOS attribute), 213
ARGUMENTS (robot.parsing.lexer.tokens.Token attribute), 212
arguments (robot.running.userkeywordrunner.EmbeddedArgumentsHandler), 307
arguments (robot.running.userkeywordrunner.UserKeywordRunner), 307
ArgumentsHandler (class in robot.result.xmlelementhandlers), 273
ArgumentSpec (class in robot.running.arguments.argumentspec), 277
ArgumentValidator (class in robot.running.arguments.argumentvalidator), 278
ArgumentValidator (class in robot.tidy), 347
as_dict() (robot.variables.scopes.GlobalVariables), 333
as_dict() (robot.variables.scopes.VariableScopes), 333
as_dict() (robot.variables.store.VariableStore method), 334
aspect() (robot.libraries.dialogs_py.InputDialog method), 119
aspect() (robot.libraries.dialogs_py.MessageDialog method), 106
aspect() (robot.libraries.dialogs_py.MultilineSelectionDialog method), 119
aspect() (robot.libraries.dialogs_py.MultilineSelectionDialog method), 145
aspect() (robot.libraries.dialogs_py.PassFailDialog method), 158
aspect() (robot.libraries.dialogs_py.SelectionDialog method), 132
assert_almost_equal() (in module robot.utils.asserts), 313
assert_equal() (in module robot.utils.asserts), 313
assert_false() (in module robot.utils.asserts), 313
assert_has_content() (robot.utils.restreader.CaptureRobotData method), 324
assert_none() (in module robot.utils.asserts), 313
assert_not_almost_equal() (in module robot.utils.asserts), 314
assert_not_equal() (in module robot.utils.asserts), 313
assert_not_none() (in module robot.utils.asserts), 313
assert_raises() (in module robot.utils.asserts), 313
assert_raises_with_msg() (in module robot.utils.asserts), 313
assert_true() (in module robot.utils.asserts), 313
assign (robot.model.keyword.Keyword attribute), 175
ASSIGN (robot.parsing.lexer.tokens.EOS attribute), 213
ASSIGN (robot.parsing.lexer.tokens.Token attribute), 212
assign (robot.running.model.statements.KeywordCallAttribute), 233
assign (robot.result.model.Keyword attribute), 260
assign() (robot.variablesassigner.VariableAssigner method), 329
assigner() (robot.variablesassigner.VariableAssignment method), 329
AssignmentHandler (class in robot.result.xmlelementhandlers), 273
AssignmentValidator (class in robot.variables.assigner), 329
AssignVarHandler (class in robot.result.xmlelementhandlers), 273
attribute_escape() (robot.utils.escapeutils) method), 320
attributes() (robot.libraries.dialogs_py.InputDialog method), 119
attributes() (robot.libraries.dialogs_py.MessageDialog method), 106
attributes() (robot.libraries.dialogs_py.MultilineSelectionDialog method), 145
attributes() (robot.libraries.dialogs_py.PassFailDialog method), 158
attributes() (robot.libraries.dialogs_py.SelectionDialog method), 132
bbox() (robot.libraries.dialogs_py.InputDialog method), 119
bbox()  (robot.libraries.dialogs_py.MessageDialog method), 106
bbox()  (robot.libraries.dialogs_py.MultilineSelectionDialog method), 145
bbox()  (robot.libraries.dialogs_py.PassFailDialog method), 158
bbox()  (robot.libraries.dialogs_py.SelectionDialog method), 132

bindtags()  (robot.libraries.dialogs_py.InputDialog method), 120
bindtags()  (robot.libraries.dialogs_py.MessageDialog method), 107
bindtags()  (robot.libraries.dialogs_py.MultilineSelectionDialog method), 107
bindtags()  (robot.libraries.dialogs_py.PassFailDialog method), 159
bindtags()  (robot.libraries.dialogs_py.SelectionDialog method), 133

bit_length()  (robot.reporting.stringcache.StringIndex method), 242

Block (class in robot.parsing.model.blocks), 214
Body (class in robot.parsing.model.blocks), 216

BooleanConverter  (class in robot.running.arguments.typeconverters), 278
build()  (robot.libdocpkg.javabuilder.JavaDocBuilder method), 23
build()  (robot.libdocpkg.robotbuilder.LibraryDocBuilder method), 23
build()  (robot.libdocpkg.robotbuilder.ResourceDocBuilder method), 23
build()  (robot.libdocpkg.specbuilder.SpecDocBuilder method), 24
build()  (robot.parsing.suitestructure.SuiteStructureBuilder method), 237
build()  (robot.reporting.jsmodelbuilders.ErrorMessageBuilder method), 239
build()  (robot.reporting.jsmodelbuilders.ErrorsBuilder method), 239
build()  (robot.reporting.jsmodelbuilders.KeywordBuilder method), 239
build()  (robot.reporting.jsmodelbuilders.KeywordBuilder method), 239
build()  (robot.reporting.jsmodelbuilders.MessageBuilder method), 239
build()  (robot.reporting.jsmodelbuilders.StatisticsBuilder method), 239
build()  (robot.reporting.jsmodelbuilders.TestSuiteBuilder method), 239
build()  (robot.reporting.jsmodelbuilders.TestBuilder method), 239
build()  (robot.result.resultbuilder.ExecutionResultBuilder method), 266
build()  (robot.running.builder.builders.TestSuiteBuilder method), 285
build()  (robot.running.builder.builders.TestSuiteBuilder method), 284
build_from()  (robot.reporting.jsmodelbuilders.JsModelBuilder method), 239
build_keyword()  (robot.libdocpkg.robotbuilder.KeywordDocBuilder method), 24
build_keywords()  (robot.libdocpkg.robotbuilder.KeywordDocBuilder method), 24
build_suite()  (robot.running.builder.parsers.RestParser method), 24
Index

C

cache_only (robot.output.logger.Logger attribute), 201
call_method (robot.libdocpkg.javabuilder.Cleaner attribute), 23
called (robot.output.listenermethods.ListenerMethod attribute), 200
can_continue() (robot.errors.ContinueForLoop method), 339
can_continue() (robot.errors.ExceptionFailed method), 337
can_continue() (robot.errors.ExceptionFailed method), 338
can_continue() (robot.errors.ExecutionPassed method), 338
can_continue() (robot.errors.ExecutionStatus method), 337
can_continue() (robot.errors.ExitForLoop method), 339
can_continue() (robot.errors.HandlerExecutionFailed method), 337

can_continue() (robot.errors.PassExecution method), 338

can_continue() (robot.errors.ReturnFromKeyword method), 339

built_suite () (robot.running.builder.parsers.RobotParser method), 285

BuiltIn (class in robot.libdocpkg.BuiltIn), 24

by_method_name() (robot.output.listenerarguments.EndKeywordArguments method), 199
by_method_name() (robot.output.listenerarguments.EndTestArguments method), 199
by_method_name() (robot.output.listenerarguments.EndSuiteArguments method), 199
by_method_name() (robot.output.listenerarguments.MessageArguments method), 199
by_method_name() (robot.output.listenerarguments.ListenerArguments method), 199
by_method_name() (robot.output.listenerarguments.StartKeywordArguments method), 199
by_method_name() (robot.output.listenerarguments.StartSuiteArguments method), 199
by_method_name() (robot.output.listenerarguments.StartTestArguments method), 199

ByteArrayConverter (class in robot.running.arguments.typeconverters), 280
BytesConverter (class in robot.running.arguments.typeconverters), 280

C

C

C

C

C
attribute), 215

col_offset (robot.parsing.model.blocks.Section attribute), 215
col_offset (robot.parsing.model.statements.Setup attribute), 220

col_offset (robot.parsing.model.statements.Statement attribute), 218
col_offset (robot.parsing.model.statements.SingleValue attribute), 230

col_offset (robot.parsing.model.blocks.TestCase attribute), 216
col_offset (robot.parsing.model.statements.TestCaseSectionHeader attribute), 217

col_offset (robot.parsing.model.blocks.VariableSection attribute), 215
col_offset (robot.parsing.model.statements.VariableSectionHeader attribute), 226

col_offset (robot.parsing.model.statements.Arguments attribute), 232
col_offset (robot.parsing.model.statements.Arguments attribute), 231

col_offset (robot.parsing.model.statements.Comment attribute), 235
col_offset (robot.parsing.model.statements.CommentSectionHeader attribute), 230

col_offset (robot.parsing.model.statements.DefaultTags attribute), 225
col_offset (robot.parsing.model.statements.DefaultTags attribute), 224

col_offset (robot.parsing.model.statements.Documentation attribute), 232
col_offset (robot.parsing.model.statements.DocumentationOrMetadata attribute), 231

col_offset (robot.parsing.model.statements.EmptyLine attribute), 215
col_offset (robot.parsing.model.statements.EmptyLine attribute), 214

col_offset (robot.parsing.model.statements.End attribute), 236
col_offset (robot.parsing.model.statements.End attribute), 227

col_offset (robot.parsing.model.statements.TestCase attribute), 235
col_offset (robot.parsing.model.statements.TestCase attribute), 227

col_offset (robot.parsing.model.statements.Error attribute), 236
col_offset (robot.parsing.model.statements.Error attribute), 228

col_offset (robot.parsing.model.statements.Fixture attribute), 219
col_offset (robot.parsing.model.statements.Fixture attribute), 228

col_offset (robot.parsing.model.statements.ForLoopHeader attribute), 225
col_offset (robot.parsing.model.statements.ForLoopHeader attribute), 224

col_offset (robot.parsing.model.statements.LibraryImport attribute), 234
col_offset (robot.parsing.model.statements.LibraryImport attribute), 229

col_offset (robot.parsing.model.statements.MessageDialog attribute), 229
col_offset (robot.parsing.model.statements.MessageDialog attribute), 228

col_offset (robot.parsing.model.statements.Return attribute), 233
col_offset (robot.parsing.model.statements.Return attribute), 232

col_offset (robot.parsing.model.statements.Setup attribute), 222
col_offset (robot.parsing.model.statements.Setup attribute), 221

col_offset (robot.parsing.model.statements.TestTemplate attribute), 231
col_offset (robot.parsing.model.statements.TestTemplate attribute), 230

col_offset (robot.parsing.model.statements.Teardown attribute), 230
col_offset (robot.parsing.model.statements.Teardown attribute), 229

col_offset (robot.parsing.model.statements.Timeout attribute), 228
col_offset (robot.parsing.model.statements.Timeout attribute), 227

col_offset (robot.parsing.model.statements.Variable attribute), 226
col_offset (robot.parsing.model.statements.Variable attribute), 225

col_offset (robot.parsing.model.statements.VariableSectionHeader attribute), 230
col_offset (robot.parsing.model.statements.VariableSectionHeader attribute), 229

col_offset (robot.parsing.model.statements.VariableSectionHeader attribute), 231
col_offset (robot.parsing.model.statements.VariableSectionHeader attribute), 228

col_offset (robot.parsing.model.statements.VariableSections (class in robot.libraries.Collections), 48
colormapwindows()
conjugate() (robot.reporting.stringcache.StringIndex method), 243
ConnectionCache (class in robot.utils.connectioncache), 314
console() (in module robot.api.logger), 15
console() (in module robot.output.librarylogger), 198
console() (robot.libdoc.LibDoc method), 340
console() (robot.rebot.Rebot method), 342
console() (robot.run.RobotFramework method), 343
console() (robot.testdoc.TestDoc method), 345
console() (robot.tidy.TidyCommandLine method), 347
console() (robot.utils.application.Application method), 310
console_colors (robot.conf.settings.RebotSettings attribute), 19
console_colors (robot.conf.settings.RobotSettings attribute), 18
console_decode() (in module robot.utils.encoding), 316
console_encode() (in module robot.utils.encoding), 316
console_markers (robot.conf.settings.RobotSettings attribute), 18
console_output_config (robot.conf.settings.RebotSettings attribute), 19
console_output_config (robot.conf.settings.RobotSettings attribute), 18
canvas_type (robot.conf.settings.RobotSettings attribute), 18
canvas_width (robot.conf.settings.RobotSettings attribute), 18
ConsoleOutput () (in module robot.output.console), 194
ConsoleViewer (class in robot.libdocpkg.consoleviewer), 22
contains_var() (in module robot.variables.isvar), 331
content() (robot.utils.markupwriters.HtmlWriter method), 321
content() (robot.utils.markupwriters.NullMarkupWriter method), 321
content() (robot.utils.markupwriters.XmlWriter method), 321
CONTINUATION (robot.parsing.lexer.tokens.EOS attribute), 213
CONTINUATION (robot.parsing.lexer.tokens.Token attribute), 212
continue_for_loop() (robot.libraries.BuiltIn.BuiltIn method), 28
continue_for_loop_if() (robot.libraries.BuiltIn.BuiltIn method), 28
continue_on_failure (robot.errors.ContinueForLoop attribute), 339
continue_on_failure (robot.errors.ExecutionFailed attribute), 337
continue_on_failure (robot.errors.ExecutionFailures attribute), 338
continue_on_failure (robot.errors.ExecutionPassed attribute), 338
continue_on_failure (robot.errors.ExecutionStatus attribute), 337
continue_on_failure (robot.errors.ExitForLoop attribute), 339
continue_on_failure (robot.errors.HandlerExecutionFailed attribute), 337
continue_on_failure (robot.errors.PassExecution attribute), 338
continue_on_failure (robot.errors.ReturnFromKeyword attribute), 340
continue_on_failure (robot.errors.UserKeywordExecutionFailed attribute), 338
ContinueForLoop, 339
convert() (robot.libdocpkg.htmlwriter.JsonConverter method), 22
convert() (robot.running.arguments.argumentconverter.ArgumentConverter method), 276
convert() (robot.running.arguments.typeconverters.BooleanConverter method), 278
convert() (robot.running.arguments.typeconverters.ByteArrayConverter method), 280
convert() (robot.running.arguments.typeconverters.BytesConverter method), 280
convert() (robot.running.arguments.typeconverters.DateConverter method), 281
convert() (robot.running.arguments.typeconverters.DateTimeConverter method), 281
convert() (robot.running.arguments.typeconverters.DecimalConverter method), 279
convert() (robot.running.arguments.typeconverters.DictionaryConverter method), 283
convert() (robot.running.arguments.typeconverters.EnumConverter method), 282
convert() (robot.running.arguments.typeconverters.FloatConverter method), 279
convert() (robot.running.arguments.typeconverters.FrozenSetConverter method), 284
convert() (robot.running.arguments.typeconverters.IntegerConverter method), 280
method), 279
convert() (robot.running.arguments.typeconverters.ListConverter, method), 282
(convert() (robot.libraries.BuiltIn.BuiltIn method), 282
convert() (robot.running.arguments.typeconverters.NoneConverter, method), 282
convert() (robot.running.arguments.typeconverters.SetConverter, method), 283
(convert() (robot.libraries.BuiltIn.BuiltIn method), 283
convert() (robot.running.arguments.typeconverters.TimeDeltaConverter, dictionary(), method), 281
(convert() (robot.libraries.Collections.Collections method), 281
convert() (robot.running.arguments.typeconverters.TupleConverter, method), 50
(convert() (robot.libraries.BuiltIn.BuiltIn method), 50
convert() (robot.running.arguments.typeconverters.TypeConverter, method), 278
(convert() (robot.libraries.BuiltIn.BuiltIn method), 278
convert() (robot.testdoc.JsonConverter method), 346
(robot.libraries.BuiltIn.BuiltIn method), 284
(robot.libraries.BuiltIn.BuiltIn method), 29
(robot.libraries.BuiltIn.BuiltIn method), 28
(robot.libraries.BuiltIn.BuiltIn method), 28
(robot.libraries.String.String method), 80
(robot.libraries.String.String method), 80
(robot.libraries.Collections.Collections method), 282
(robot.libraries.Collections.Collections method), 282
(robot.libraries.Collections.Collections method), 282
(robot.libraries.Collections.Collections method), 282
(robot.libraries.Collections.Collections method), 282
(robot.libraries.BuiltIn.BuiltIn method), 282
(robot.libraries.BuiltIn.BuiltIn method), 282
(robot.libraries.BuiltIn.BuiltIn method), 282
(robot.libraries.BuiltIn.BuiltIn method), 282
(robot.libraries.BuiltIn.BuiltIn method), 282
(robot.libraries.BuiltIn.BuiltIn method), 282
(robot.libraries.BuiltIn.BuiltIn method), 282
(robot.libraries.BuiltIn.BuiltIn method), 282
(robot.libraries.BuiltIn.BuiltIn method), 282
(robot.libraries.BuiltIn.BuiltIn method), 282
(robot.libraries.BuiltIn.BuiltIn method), 282
(robot.libraries.BuiltIn.BuiltIn method), 282
(robot.libraries.BuiltIn.BuiltIn method), 282
(robot.libraries.BuiltIn.BuiltIn method), 282
(robot.libraries.BuiltIn.BuiltIn method), 282
(robot.libraries.BuiltIn.BuiltIn method), 282
(robot.libraries.BuiltIn.BuiltIn method), 282
(robot.libraries.BuiltIn.BuiltIn method), 282
(robot.libraries.BuiltIn.BuiltIn method), 282
(robot.libraries.BuiltIn.BuiltIn method), 282
(robot.libraries.BuiltIn.BuiltIn method), 282
(robot.libraries.BuiltIn.BuiltIn method), 282
(robot.libraries.BuiltIn.BuiltIn method), 282
(robot.libraries.BuiltIn.BuiltIn method), 282
(robot.libraries.BuiltIn.BuiltIn method), 282
(robot.libraries.BuiltIn.BuiltIn method), 282
(robot.libraries.BuiltIn.BuiltIn method), 282
(robot.libraries.BuiltIn.BuiltIn method), 282
(robot.libraries.BuiltIn.BuiltIn method), 282
(robot.libraries.BuiltIn.BuiltIn method), 282
(robot.libraries.BuiltIn.BuiltIn method), 282
(robot.libraries.BuiltIn.BuiltIn method), 282
(robot.libraries.BuiltIn.BuiltIn method), 282
(robot.libraries.BuiltIn.BuiltIn method), 282
(robot.libraries.BuiltIn.BuiltIn method), 282
(robot.libraries.BuiltIn.BuiltIn method), 282
(robot.libraries.BuiltIn.BuiltIn method), 282
(robot.libraries.BuiltIn.BuiltIn method), 282
(robot.libraries.BuiltIn.BuiltIn method), 282
(robot.libraries.BuiltIn.BuiltIn method), 282
(robot.libraries.BuiltIn.BuiltIn method), 282
(robot.libraries.BuiltIn.BuiltIn method), 282
(robot.libraries.BuiltIn.BuiltIn method), 282
(robot.libraries.BuiltIn.BuiltIn method), 282
(robot.libraries.BuiltIn.BuiltIn method), 282
(robot.libraries.BuiltIn.BuiltIn method), 282
(robot.libraries.BuiltIn.BuiltIn method), 282
converter_for() (robot.running.arguments.typeconverters.ListConverter, 282)
count() (robot.model.testsuite.TestSuites method, 188)
count() (robot.model.itemlist.ItemList method, 175)
count() (robot.model.keyword.Keywords method, 177)
count() (robot.model.message.Messages method, 178)
count() (robot.model.testcase.TestCases method, 188)
copy() (robot.model.keyword.Keyword method, 176)
copy() (robot.model.message.Message method, 178)
copy() (robot.model.metadata.Metadata method, 179)
copy() (robot.model.modelobject.ModelObject method, 179)
copy() (robot.model.testcase.TestCase method, 188)
copy() (robot.model.testsuite.TestSuite method, 190)
copy() (robot.output.loggerhelper.Message method, 202)
copy() (robot.result.model.Keyword method, 260)
copy() (robot.result.model.Message method, 259)
copy() (robot.result.model.TestCase method, 261)
copy() (robot.result.model.TestSuite method, 264)
copy() (robot.running.model.ForLoop method, 295)
copy() (robot.running.model.Keyword method, 293)
copy() (robot.running.model.TestCase method, 296)
copy() (robot.running.model.TestSuite method, 298)
copy() (robot.utils.dotdict.DotDict method, 315)
copy() (robot.variables.normalizing.NormalizedDict method, 323)
copy() (robot.variables.scopes.GlobalVariables method, 333)
copy() (robot.variables.variables.Variables method, 335)
copy_dictionary() (robot.libraries.Collections.Collections method, 50)
copy_element() (robot.libraries.XML.XML method, 104)
copy_files() (robot.libraries.OperatingSystem.OperatingSystem method, 66)
copy_list() (robot.libraries.Collections.Collections method, 50)
count() (robot.model.itemlist.ItemList method, 175)
count() (robot.model.keyword.Keywords method, 177)
count() (robot.model.message.Messages method, 178)
data_tokens (robot.parsing.model.statements.TestCaseSectionHeader attribute), 221
deepcopy () (robot.result.model.Keyword method), 202
data_tokens (robot.parsing.model.statements.TestSetup attribute), 227
deepcopy () (robot.result.model.Message method), 260
data_tokens (robot.parsing.model.statements.TestTeardown attribute), 227
deepcopy () (robot.result.model.TestCase method), 259
data_tokens (robot.parsing.model.statements.TestTemplate attribute), 228
deepcopy () (robot.result.model.TestSuite method), 262
data_tokens (robot.parsing.model.statements.Timeout attribute), 228
deepcopy () (robot.running.model.Keyword method), 264
data_tokens (robot.parsing.model.statements.Variable attribute), 229
deepcopy () (robot.running.model.TestCase method), 293
data_tokens (robot.parsing.model.statements.VariableSectionHeader attribute), 229
DEFAULT_TAGS (robot.parsing.lexer.tokens.EOS attribute), 220
data_tokens (robot.parsing.model.statements.VariablesImport attribute), 223
DEFAULT_TAGS (robot.parsing.lexer.tokens.Token attribute), 298
data_tokens (robot.parsing.model.statements.TestTimeout attribute), 232
DefaultLogger (class in robot.utils.application), 310
data_tokens (robot.parsing.model.statements.TestTemplateSectionHeader attribute), 243
default_value (class in robot.running.arguments.argumentmapper), 276
data_tokens (robot.parsing.model.statements.TestTemplateSectionHeader attribute), 258
defeature (class in robot.libraries.dialogs_py.InputDialog), 121
data_tokens (robot.parsing.model.statements.VariableSectionHeader attribute), 260
defeature (class in robot.libraries.dialogs_py.MessageDialog method), 108
data_tokens (robot.parsing.model.statements.VariableSectionHeader attribute), 325
defeature (class in robot.libraries.dialogs_py.SelectionDialog method), 147
data_tokens (robot.parsing.model.statements.VariableSectionHeader attribute), 340
defeature (class in robot.libraries.dialogs_py.MultipleSelectionDialog method), 147
data_tokens (robot.parsing.model.statements.VariableSectionHeader attribute), 512
defeature (class in robot.libraries.dialogs_py.PassFailDialog method), 160
data_tokens (robot.parsing.model.statements.VariableSectionHeader attribute), 610
defeature (class in robot.libraries.dialogs_py.SelectionDialog method), 134
data_tokens (robot.parsing.model.statements.VariableSectionHeader attribute), 634
del_env_var () (in module robot.utils.robotenv), 325
data_tokens (robot.parsing.model.statements.VariableSectionHeader attribute), 743
delayed_logging (robot.output.logger.Logger attribute), 201
data_tokens (robot.parsing.model.statements.VariableSectionHeader attribute), 758
delecommand () (robot.libraries.dialogs_py.InputDialog method), 121
data_tokens (robot.parsing.model.statements.VariableSectionHeader attribute), 772
delecommand () (robot.libraries.dialogs_py.MessageDialog method), 108
data_tokens (robot.parsing.model.statements.VariableSectionHeader attribute), 787
delecommand () (robot.libraries.dialogs_py.SelectionDialog method), 147
data_tokens (robot.utils.restreader.CaptureRobotData method), 324
delecommand () (robot.libraries.dialogs_py.PassFailDialog method), 160
data_tokens (robot.utils.restreader.CaptureRobotData method), 340
delecommand () (robot.libraries.dialogs_py.SelectionDialog method), 134
data_tokens (robot.utils.restreader.CaptureRobotData method), 512
delecommand () (robot.libraries.dialogs_py.SelectionDialog method), 147
data_tokens (robot.utils.restreader.CaptureRobotData method), 527
denominator (robot.reporting.stringcache.StringIndex attribute), 243
data_tokens (robot.utils.restreader.CaptureRobotData method), 743
destroy () (robot.libraries.dialogs_py.InputDialog method), 121
destroy() (robot.libraries.dialogs_py.MessageDialog method), 108

destroy() (robot.libraries.dialogs_py.MultipleSelectionDialog method), 147
destroy() (robot.libraries.dialogs_py.PassFailDialog method), 160
destroy() (robot.libraries.dialogs_py.SelectionDialog method), 134

dictDumper (class in robot.htmldata.jsonwriter), 21
dictionaries_should_be_equal() (robot.libraries.Collections.Collections method), 50
dictionary_should_contain_item() (robot.libraries.Collections.Collections method), 51
dictionary_should_contain_key() (robot.libraries.Collections.Collections method), 51
dictionary_should_contain_sub_dictionary() (robot.libraries.Collections.Collections method), 51
dictionary_should_contain_value() (robot.libraries.Collections.Collections method), 51
dictionary_should_not_contain_key() (robot.libraries.Collections.Collections method), 51
dictionary_should_not_contain_value() (robot.libraries.Collections.Collections method), 51

DictionaryConverter (class in robot.running.arguments.typeconverters), 283

DictToKwargs (class in robot.running.arguments.argumentresolver), 277

DictVariableTableValue (class in robot.variables.tablesetter), 335
directive_error() (robot.utils.restreader.CaptureRobotData method), 324
directory (robot.running.model.Import attribute), 300
directory() (robot.tidy.Tidy method), 347
directory_should_be_empty() (robot.libraries.OperatingSystem.OperatingSystem method), 64
directory_should_exist() (robot.libraries.OperatingSystem.OperatingSystem method), 64
directory_should_not_be_empty() (robot.libraries.OperatingSystem.OperatingSystem method), 64
directory_should_not_exist()
dont_continue (robot.errors.ExitForLoop attribute), 339
dont_continue (robot.errors.HandlerExecutionFailed attribute), 337
dont_continue (robot.errors.PassExecution attribute), 338
dont_continue (robot.errors.ReturnFromKeyword attribute), 340
dont_continue (robot.errors.UserKeywordExecutionFailed attribute), 338
DosHighlighter (class in robot.output.console.highlighting), 196
DotDict (class in robot.utils.dotdict), 315
DottedImporter (class in robot.utils.importer), 320
DottedOutput (class in robot.output.console.dotted), 195
dry_run (robot.conf.settings.RobotSettings attribute), 18
dry_run () (robot.running.librarykeywordrunner.EmbeddedArgumentsRunner method), 291
dry_run () (robot.running.librarykeywordrunner.LibraryKeywordRunner method), 291
dry_run () (robot.running.librarykeywordrunner.RunKeywordRunner method), 321
dry_run () (robot.running.usererrorhandler.UserErrorHandler method), 307
dry_run () (robot.running.userkeywordrunner.EmbeddedArgumentsRunner method), 307
dry_run () (robot.running.userkeywordrunner.UserKeywordRunner (robot.libraries.XML.XML method)), 307
dump () (robot.htmldata.jsonwriter.DictDumper method), 21
dump () (robot.htmldata.jsonwriter.IntegerDumper method), 21
dump () (robot.htmldata.jsonwriter.JsonDumper method), 21
dump () (robot.htmldata.jsonwriter.MappingDumper method), 21
dump () (robot.htmldata.jsonwriter.NoneDumper method), 21
dump () (robot.htmldata.jsonwriter.StringDumper method), 21
dump () (robot.htmldata.jsonwriter.TupleListDumper method), 21
dump () (robot.reporting.stringcache.StringCache method), 243
DynamicArgumentParser (class in robot.running.arguments.argumentparser), 276
DynamicHandler () (in module robot.running.handlers), 290

earlier_failures (robot.errors.ContinueForLoop attribute), 339
earlier_failures (robot.errors.ExecutionPassed attribute), 338
earlier_failures (robot.errors.ExitForLoop attribute), 339
earlier_failures (robot.errors.PassExecution attribute), 339
earlier_failures (robot.errors.ReturnFromKeyword attribute), 340
earlier_failures (robot.errors.UserKeywordExecutionFailed attribute), 338
elapsed (robot.model.stats.Stat attribute), 182
elapsed (robot.model.stats.SuiteStat attribute), 183
elapsed_time_to_string() (in module robot.utils.robottime), 327
elapsedtime (robot.result.model.Keyword attribute), 260
elapsedtime (robot.result.model.TestCase attribute), 261
elapsedtime (robot.result.model.TestSuite attribute), 261
ElementComparator (class in robot.libraries.XML.XML), 105
ElementFinder (class in robot.libraries.XML.XML), 105
elements_should_be_equal() (robot.libraries.XML.XML method), 102
elements_should_match() (robot.libraries.XML.XML method), 102
ElementTo_string() (robot.libraries.XML.XML method), 105
ElementComparer (in module robot.libraries.XML.XML), 105
EmbeddedArgumentParser (class in robot.running.arguments.embedded), 278
EmbeddedArguments (class in robot.running.arguments.embedded), 278
EmbeddedArgumentshandler (class in robot.running.handlers), 290
Index
end_keyword() (robot.conf.gatherfailed.GatherFailedSuites, 17)
end_keyword() (robot.conf.gatherfailed.GatherFailedTests, 16)
end_keyword() (robot.model.configurer.SuiteConfigurer, 171)
end_keyword() (robot.model.filter.EmptySuiteRemover, 174)
end_keyword() (robot.model.modifier.ModelModifier, 180)
end_keyword() (robot.model.statistics.StatisticsBuilder, 181)
end_keyword() (robot.model.tagsetter.TagSetter, 186)
end_keyword() (robot.model.totalstatistics.TotalStatisticsBuilder, 191)
end_keyword() (robot.model.visitor.SuiteVisitor, 194)
end_keyword() (robot.output.console.dotted.StatusReporter, 195)
end_keyword() (robot.output.console.verbose.VerboseOutput, 197)
end_keyword() (robot.output.filelogger.FileLogger, 201)
end_keyword() (robot.output.output.Output, 203)
end_keyword() (robot.output.xmllogger.XmlLogger, 205)
end_keyword() (robot.reporting.outputwriter.OutputWriter, 241)
end_keyword() (robot.reporting.xunitwriter.XUnitFileWriter, 244)
end_keyword() (robot.result.configurer.SuiteConfigurer, 246)
end_keyword() (robot.result.keywordremover.AllKeywordsRemover, 250)
end_keyword() (robot.result.keywordremover.ByNameKeywordRemover, 252)
end_keyword() (robot.result.keywordremover.ByTagKeywordRemover, 253)
end_keyword() (robot.result.keywordremover.ForLoopItemsRemover, 254)
end_keyword() (robot.result.keywordremover.PassedKeywordRemover, 255)
end_keyword() (robot.result.keywordremover.WaitUntilKeywordSucceedsRemover, 256)
end_keyword() (robot.result.messagefilter.MessageFilter, 257)
end_keyword() (robot.result.resultbuilder.RemoveKeywords, 266)
end_keyword() (robot.result.suiteteardownfailed.SuiteTeardownFailed, 268)
end_keyword() (robot.running.randomizer.Randomizer, 270)
end_keyword() (robot.running.runner.Runner, 302)
end_keyword() (robot.variables.scopes.SetVariables, 333)
end_keyword() (robot.variables.scopes.VariableScopes, 332)
end_keyword() (robot.parsing.model.blocks.Block, 215)
end_keyword() (robot.parsing.model.blocks.CommentSection, 216)
end_keyword() (robot.parsing.model.blocks.CommentSection, 215)
end_keyword() (robot.parsing.model.blocks.Keyword, 216)
end_keyword() (robot.parsing.model.blocks.Keyword, 215)
end_keyword() (robot.parsing.model.blocks.ForLoop, 216)
end_keyword() (robot.parsing.model.blocks.TestCase, 216)
end_keyword() (robot.parsing.model.blocks.KeywordSection, 216)
end_keyword() (robot.parsing.model.blocks.TestCaseSection, 215)
end_keyword() (robot.parsing.model.blocks.SettingSection, 215)
end_keyword() (robot.parsing.model.blocks.DefaultTags, 216)
end_keyword() (robot.parsing.model.blocks.VariableSection, 215)
end_keyword() (robot.parsing.model.blocks.DocumentationSection, 215)
end_keyword() (robot.parsing.model.blocks.DocumentationOrMetadata, 215)
end_lineno (robot.parsing.model.statements.EmptyLine attribute), 236
end_lineno (robot.parsing.model.statements.End attribute), 235
end_lineno (robot.parsing.model.statements.Error attribute), 236
end_lineno (robot.parsing.model.statements.Fixture attribute), 219
end_lineno (robot.parsing.model.statements.ForceTags attribute), 225
end_lineno (robot.parsing.model.statements.ForLoopHeader attribute), 234
end_lineno (robot.parsing.model.statements.KeywordCall attribute), 229
end_lineno (robot.parsing.model.statements.KeywordName attribute), 230
end_lineno (robot.parsing.model.statements.LibraryImport attribute), 223
end_lineno (robot.parsing.model.statements.LibraryImport.d_message) (robot.conf.gatherfailed.GatherFailedSuites method), 17
end_lineno (robot.parsing.model.statements.MetaData attribute), 224
end_lineno (robot.parsing.model.statements.MultiValue attribute), 200
end_lineno (robot.parsing.model.statements.ResourceImport attribute), 222
end_lineno (robot.parsing.model.statements.Return attribute), 233
end_lineno (robot.parsing.model.statements.Setup attribute), 206
end_lineno (robot.parsing.model.statements.SetupSectionHeader attribute), 230
end_lineno (robot.parsing.model.statements.SetupSectionHeader.d_message) (robot.model.filter.Filter method), 174
end_lineno (robot.parsing.model.statements.Statement attribute), 217
end_lineno (robot.parsing.model.statements.StatementSectionHeader attribute), 220
end_lineno (robot.parsing.model.statements.TestCaseSectionHeader attribute), 221
end_lineno (robot.parsing.model.statements_TestCaseSectionSectionHeader attribute), 218
end_lineno (robot.parsing.model.statements_TestCaseSectionSectionHeader.d_message) (robot.model.totalstatistics.TotalStatisticsBuilder method), 191
end_lineno (robot.parsing.model.statements_TestCaseSectionSectionHeader.d_message) (robot.output.console.dotted.StatusReporter method), 194
end_lineno (robot.parsing.model.statements_TestCaseSectionSectionHeader.d_message) (robot.output.xmllogger.XmlLogger method), 195
end_lineno (robot.parsing.model.statements_TestCaseSectionSectionHeader.d_message) (robot.output.console.dotted.StatusReporter method), 196
end_lineno (robot.parsing.model.statements_TestCaseSectionSectionHeader.d_message) (robot.output.xmllogger.XmlLogger method), 200
end_lineno (robot.parsing.model.statements_TestCaseSectionSectionHeader.d_message) (robot.reporting.outputwriter.OutputStream method), 241
end_lineno (robot.parsing.model.statements_TestCaseSectionSectionHeader.d_message) (robot.reporting.xmlwriter.XUnitFileWriter method), 244
end_lineno (robot.parsing.model.statements_TestCaseSectionSectionHeader.d_message) (robot.result.configurer.SuiteConfigurer method), 246
end_lineno (robot.parsing.model.statements_TestCaseSectionSectionHeader.d_message) (robot.result.keywordremover.AllKeywordsRemover method), 250
end_lineno (robot.parsing.model.statements_TestCaseSectionSectionHeader.d_message) (robot.result.keywordremover.ByNameKeywordRemover method), 252
end_lineno (robot.parsing.model.statements_TestCaseSectionSectionHeader.d_message) (robot.result.keywordremover.ByTagKeywordRemover method), 253
end_message() (robot.result.keywordremover.ForLoopItemsRemover method), 254
end_message() (robot.result.keywordremover.PassedKeywordRemover method), 251
end_message() (robot.result.keywordremover.WaitUntilKeywordSucceedsRemover method), 255
end_message() (robot.result.merger.Merger method), 257
end_message() (robot.result.messagefilter.MessageFilter method), 258
end_message() (robot.result.resultbuilder.RemoveKeywords method), 266
end_message() (robot.result.suiteardownfailed.SuiteTeardownFailed method), 267
end_message() (robot.result.suiteardownfailed.SuiteTeardownFailedHandler method), 270
end_message() (robot.running.randomizer.Randomizer method), 302
end_result() (robot.output.xmllogger.XmlLogger method), 206
end_result() (robot.reporting.outputwriter.OutputWriter method), 240
end_result() (robot.reporting.xunitwriter.XUnitFileWriter method), 244
end_result() (robot.result.visitor.ResultVisitor method), 269
end_splitting() (robot.reporting.jsbuildingcontext.JsBuildingContext method), 238
end_stat() (robot.output.xmllogger.XmlLogger method), 206
end_stat() (robot.reporting.outputwriter.OutputWriter method), 241
end_stat() (robot.reporting.xunitwriter.XUnitFileWriter method), 244
end_stat() (robot.result.visitor.ResultVisitor method), 270
end_statistics() (robot.output.xmllogger.XmlLogger method), 205
end_statistics() (robot.reporting.outputwriter.OutputWriter method), 241
end_statistics() (robot.reporting.xunitwriter.XUnitFileWriter method), 244
end_statistics() (robot.result.visitor.ResultVisitor method), 269
end_suite() (robot.conf.gatherfailed.GatherFailedSuites method), 17
end_suite() (robot.conf.gatherfailed.GatherFailedTests method), 16
event_generate() (robot.libraries.dialogs_py.MessageDialog method), 108

event_generate() (robot.libraries.dialogs_py.MultipleSelectionDialog method), 148

event_generate() (robot.libraries.dialogs_py.PassFailDialog method), 161

event_generate() (robot.libraries.dialogs_py.SelectionDialog method), 135

event_info() (robot.libraries.dialogs_py.InputDialog method), 122

event_info() (robot.libraries.dialogs_py.MessageDialog method), 109

event_info() (robot.libraries.dialogs_py.MultipleSelectionDialog method), 148

event_info() (robot.libraries.dialogs_py.PassFailDialog method), 161

event_info() (robot.libraries.dialogs_py.SelectionDialog method), 135

exclude_tags (robot.model.filter.Filter attribute), 173

execute() (robot.libdoc.LibDoc method), 340

execute() (robot.rebot.Rebot method), 342

execute() (robot.run.RobotFramework method), 343

execute() (robot.running.timeoutsposix.Timeout method), 289

execute() (robot.running.timeouts.windows.Timeout method), 289

execute() (robot.tidy.TidyDoc method), 345

execute() (robot.tidy.TidyCommandLine method), 347

execute() (robot.utils.application.Application method), 310

execute_cli() (robot.libdoc.LibDoc method), 340

execute_cli() (robot.rebot.Rebot method), 342

execute_cli() (robot.run.RobotFramework method), 343

execute_cli() (robot.testdoc.TestDoc method), 345

execute_cli() (robot.tidy.TidyCommandLine method), 347

execute_cli() (robot.utils.application.Application method), 310

execute_command() (robot.libraries.Telnet.TelnetConnection method), 92

execute_manual_step() (in module robot.libraries.Dialogs), 60

ExecutionContexts (class in robot.running.context), 289

ExecutionErrors (class in robot.result.executionerrors), 247

ExecutionFailed, 337

ExecutionFailures, 337

ExecutionPassed, 338

ExecutionResult (class in robot.libraries.Process), 76

F

fail() (in module robot.utils.asserts), 313

fail() (robot.libraries.BuiltIn.BuiltIn method), 31

fail() (robot.output.filelogger.FileLogger method), 198
fail() (robot.output.logger.Logger method), 201
fail() (robot.output.loggerhelper.AbstractLogger method), 202
fail() (robot.output.output.Output method), 203
failed (robot.model.stats.Stat attribute), 182
Failure (class in robot.running.status), 304
failure_occurred() (robot.running.status.Exit method), 304
failures (robot.running.status.SuiteStatus attribute), 304
failures (robot.running.status.TestStatus attribute), 304
FATAL_ERROR (robot.parsing.lexer.tokens.EOS attribute), 213
FATAL_ERROR (robot.parsing.lexer.tokens.Token attribute), 212
feed() (robot.libraries.Telnet.TelnetTerminalEmulator method), 94
fetch_from_left() (robot.libraries.String.String method), 83
fetch_from_right() (robot.libraries.String.String method), 83
File (class in robot.parsing.model.blocks), 215
file() (robot.tidy.Tidy method), 346
file_should_be_empty() (robot.libraries.OperatingSystem.OperatingSystem ind_and_format() (robot.parsing.model.statements.ForLoopHeader method), 323
file_should_exist() (robot.libraries.OperatingSystem.OperatingSystem ind_file() (in module robot.utils.robotpath), 325
file_writer() (in module robot.utils.robotio), 325
FileContext (class in robot.parsing.lexer.context), 208
FileLogger (class in robot.output.filelogger), 198
fileno() (robot.libraries.Telnet.TelnetConnection method), 93
FileReader (class in robot.utils.filereader), 318
fill_named() (robot.running.arguments.argumentsmapper.KeywordCallTemplate method), 276
fill_positional() (robot.running.arguments.argumentsmapper.KeywordCallTemplate method), 276
fill_rawq() (robot.libraries.Telnet.TelnetConnection method), 93
Filter (class in robot.model.filter), 173
filter() (robot.model.testsuite.TestSuite method), 190
filter() (robot.output.pyloggingconf.RobotHandler method), 204
filter() (robot.result.model.TestSuite method), 264
filter() (robot.running.model.TestSuite method), 299
filter_messages() (robot.result.model.TestSuite method), 263
final_argument_whitespace (robot.utils.restreader.CaptureRobotData attribute), 324
find() (robot.utils.recommendations.RecommendationFinder method), 323
find() (robot.variables.finders.EmptyFinder class method), 331
find() (robot.variables.finders.EnvironmentFinder method), 331
find() (robot.variables.finders.InlinePythonFinder method), 331
find() (robot.variables.finders.NumberFinder method), 331
find() (robot.variables.finders.StoredFinder method), 331
find() (robot.variables.finders.VariableFinder method), 330
find_all() (robot.libraries.XML.ElementFinder method), 105
find_from() (robot.parsing.model.blocks.FirstStatementFinder class method), 217
FirstStatementFinder (class in robot.parsing.model.blocks), 217
Fixture (class in robot.parsing.model.statements), 219
fixture() (in module robot.running.builder.transformers), 286
flatten_keywords (robot.conf.settings.RebotSettings attribute), 19
flatten_keywords (robot.conf.settings.RobotSettings attribute), 18
FlattenByKeywordMatcher (class in robot.result.flattenkeywordmatcher), 250
FlattenByNamedMatcher (class in robot.result.flattenkeywordmatcher), 250
FlattenByTagMatcher (class in robot.result.flattenkeywordmatcher), 250
flavor (robot.parsing.model.blocks.ForLoop attribute), 216
flavor (robot.parsing.model.statements.ForLoopHeader attribute), 234
get_child_handler() (robot.result.xmlelementhandlers.TagHandler method), 273
get_child_handler() (robot.result.xmlelementhandlers.TagsHandler method), 273
get_child_handler() (robot.result.xmlelementhandlers.TestCaseHandler method), 271
get_child_handler() (robot.result.xmlelementhandlers.TestStatusHandler method), 272
get_child_handler() (robot.result.xmlelementhandlers.TimeoutHandler method), 273
get_combined_stats() (robot.model.tagstatistics.TagStatInfo method), 187
get_connection() (robot.utils.connectioncache.ConnectionCache method), 315
get_console_encoding() (in module robot.utils.encodingsniffer), 317
get_console_length() (in module robot.utils.text), 328
get_converter() (robot.running.arguments.typeconverters.BooleanConverter method), 279
get_converter() (robot.running.arguments.typeconverters.ByteArrayConverter method), 280
get_converter() (robot.running.arguments.typeconverters.BytesConverter method), 280
get_converter() (robot.running.arguments.typeconverters.DateConverter method), 281
get_converter() (robot.running.arguments.typeconverters.DateTimeConverter method), 281
get_converter() (robot.running.arguments.typeconverters.DecimalConverter method), 280
get_converter() (robot.running.arguments.typeconverters.DictionaryConverter method), 283
get_converter() (robot.running.arguments.typeconverters.EnumConverter method), 282
get_converter() (robot.running.arguments.typeconverters.FloatConverter method), 279
get_converter() (robot.running.arguments.typeconverters.FrozenSetConverter method), 282
get_converter() (robot.running.arguments.typeconverters.IntegerConverter method), 279
get_converter() (robot.running.arguments.typeconverters.ListConverter method), 282
get_converter() (robot.running.arguments.typeconverters.NoneConverter method), 282
get_converter() (robot.running.arguments.typeconverters.SetConverter method), 283
get_converter() (robot.running.arguments.typeconverters.TimeDeltaConverter method), 281
get_converter() (robot.running.arguments.typeconverters.TupleConverter method), 283
get_converter() (robot.running.arguments.typeconverters.TypeConverter method), 278
get_count() (robot.libraries.BuiltIn.BuiltIn method), 31
get_critical_stats() (robot.model.tagstatistics.TagStatInfo method), 187
get_current_date() (in module robot.libraries.DateTime), 57
get_data() (robot.utils.restreader.RobotDataStorage method), 324
get_dictionary_items() (robot.libraries.Collections.Collections method), 51
get_dictionary_keys() (robot.libraries.Collections.Collections method), 51
get_dictionary_values() (robot.libraries.Collections.Collections method), 51
get_doc() (robot.model.tagstatistics.TagStatInfo method), 187
get_elapsed_time() (in module robot.utils.robottime), 327
get_element() (robot.libraries.XML.XML method), 99
get_element_attribute() (robot.libraries.XML.XML method), 101
get_element_attributes() (robot.libraries.XML.XML method), 101
get_element_count() (robot.libraries.XML.XML method), 100
get_element_text() (robot.libraries.XML.XML method), 100
get_elements() (robot.libraries.XML.XML method), 99
get_elements_texts() (robot.libraries.XML.XML method), 99
get_env_var() (in module robot.utils.robotenv), 325
get_env_vars() (in module robot.utils.robotenv), 325
get_environment_variable() (robot.libraries.OperatingSystem.OperatingSystem method), 67
get_environment_variables() (robot.libraries.OperatingSystem.OperatingSystem method), 67
get_error_details() (in module robot.utils.error), 317
get_error_message() (in module robot.utils.error), 317
get_first() (robot.running.arguments.typeconverters.FrozenSetConverter method), 282
get_first() (robot.running.arguments.typeconverters.IterableConverter method), 282
get_first() (robot.running.arguments.typeconverters.ListConverter method), 282
get_first() (robot.running.arguments.typeconverters.SetConverter method), 283
get_first() (robot.running.arguments.typeconverters.TimeDeltaConverter method), 281
get_first() (robot.running.arguments.typeconverters.TupleConverter method), 283
get_first() (robot.running.arguments.typeconverters.TypeConverter method), 278
get_first() (robot.running.arguments.typeconverters.WeekConverter method), 283
get_first() (robot.running.arguments.typeconverters.YearMonthConverter method), 283
get_first() (robot.running.arguments.typeconverters.YearWeekConverter method), 283
get_matches() (robot.libraries.Collections.Collections method), 50
get_message() (robot.running.timeouts.KeywordTimeout method), 49
get_message() (robot.running.timeouts.TestTimeout method), 289
get_message() (robot.running.timeouts.KeywordTimeout method), 288
get_modified_time() (robot.libraries.OperatingSystem.OperatingSystem method), 69
get_name() (robot.output.pyloggingconf.RobotHandler method), 204
get_process_id() (robot.libraries.Process.Process method), 75
get_process_result() (robot.libraries.Process.Process method), 75
get_rebot_settings() (robot.conf.settings.RobotSettings method), 18
get_regexp_matches() (robot.libraries.String.String method), 82
get_runner() (robot.running.namespace.KeywordStore method), 301
get_runner() (robot.running.namespace.Namespace method), 301
get_selection_from_user() (in module robot.libraries.Dialogs), 60
get_selections_from_user() (in module robot.libraries.Dialogs), 60
get_slice_from_list() (robot.libraries.Collections.Collections method), 52
get_socket() (robot.libraries.Telnet.TelnetConnection method), 93
get_stat() (robot.model.tagstatistics.TagStatInfo method), 187
get_substring() (robot.libraries.String.String method), 84
get_system_encoding() (in module robot.utils.encodingsniffer), 317
get_time() (in module robot.utils.robottime), 326
get_time() (robot.libraries.BuiltIn.Builtin method), 32
get_timestamp() (in module robot.utils.robottime), 326
get_timestamp() (robot.utils.robottime.TimestampCache method), 327
get_token() (robot.parsing.model.statements.Arguments method), 232
get_token() (robot.parsing.model.statements.CommentSectionHeader method), 222
get_token() (robot.parsing.model.statements.DefaultTags method), 225
get_token() (robot.parsing.model.statements.Documentation method), 224
get_token() (robot.parsing.model.statements.DocumentationOrMetadata method), 218
get_token() (robot.parsing.model.statements.EmptyLine method), 236
get_token() (robot.parsing.model.statements.End method), 235
get_token() (robot.parsing.model.statements.Error method), 236
get_token() (robot.parsing.model.statements.Fixture method), 219
get_token() (robot.parsing.model.statements.ForceTags method), 225
get_token() (robot.parsing.model.statements.ForLoopHeader method), 234
get_token() (robot.parsing.model.statements.KeywordCall method), 233
get_token() (robot.parsing.model.statements.KeywordName method), 230
get_token() (robot.parsing.model.statements.LibraryImport method), 223
get_token() (robot.parsing.model.statements.Metadata method), 224
get_token() (robot.parsing.model.statements.MultiValue method), 219
get_token() (robot.parsing.model.statements.ResourceImport method), 223
get_token() (robot.parsing.model.statements.Return method), 233
get_token() (robot.parsing.model.statements.SectionHeader method), 220
get_token() (robot.parsing.model.statements.SettingSectionHeader method), 220
get_token() (robot.parsing.model.statements.Setup method), 230
get_token() (robot.parsing.model.statements.TagStatInfo method), 231
get_token() (robot.parsing.model.statements.SyntaxError method), 236
get_token() (robot.parsing.model.statements.Tags method), 231
get_token() (robot.parsing.model.statements.Teardown method), 226
Index 387
Index 391
grid_columnconfigure()
(robot.libraries.dialogs_py.PassFailDialog method), 162
grid_columnconfigure()
(robot.libraries.dialogs_py.SelectionDialog method), 136
grid_location()
(robot.libraries.dialogs_py.InputDialog method), 123
grid_location()
(robot.libraries.dialogs_py.MessageDialog method), 110
grid_location()
(robot.libraries.dialogs_py.MultipleSelectionDialog method), 149
gird_location()
(robot.libraries.dialogs_py.PassFailDialog method), 162
gird_location()
(robot.libraries.dialogs_py.SelectionDialog method), 136
grid_propagate()
(robot.libraries.dialogs_py.InputDialog method), 123
grid_propagate()
(robot.libraries.dialogs_py.MessageDialog method), 110
gird_propagate()
(robot.libraries.dialogs_py.MultipleSelectionDialog method), 149
gird_propagate()
(robot.libraries.dialogs_py.PassFailDialog method), 162
gird_propagate()
(robot.libraries.dialogs_py.SelectionDialog method), 136
gird_rowconfigure()
(robot.libraries.dialogs_py.InputDialog method), 123
gird_rowconfigure()
(robot.libraries.dialogs_py.MessageDialog method), 110
gird_rowconfigure()
(robot.libraries.dialogs_py.MultipleSelectionDialog method), 149
gird_rowconfigure()
(robot.libraries.dialogs_py.PassFailDialog method), 162
gird_rowconfigure()
(robot.libraries.dialogs_py.SelectionDialog method), 136
gird_size()
(robot.libraries.dialogs_py.InputDialog method), 123
gird_size()
(robot.libraries.dialogs_py.MessageDialog method), 110
gird_size()
(robot.libraries.dialogs_py.MultipleSelectionDialog method), 149
gird_size()
(robot.libraries.dialogs_py.PassFailDialog method), 162
gird_size()
(robot.libraries.dialogs_py.SelectionDialog method), 136
gird_slaves()
(robot.libraries.dialogs_py.InputDialog method), 123
gird_slaves()
(robot.libraries.dialogs_py.MessageDialog method), 110
gird_slaves()
(robot.libraries.dialogs_py.MultipleSelectionDialog method), 149
grid_slaves()
(robot.libraries.dialogs_py.PassFailDialog method), 162
gird_slaves()
(robot.libraries.dialogs_py.SelectionDialog method), 136
grid_slaves()
(robot.libraries.dialogs_py.MessageDialog method), 110
grid_slaves()
(robot.libraries.dialogs_py.MultipleSelectionDialog method), 149
gird_slaves()
(robot.libraries.dialogs_py.PassFailDialog method), 162
gird_slaves()
(robot.libraries.dialogs_py.SelectionDialog method), 136
grid_slaves()
(robot.libraries.dialogs_py.MessageDialog method), 110
grid_slaves()
(robot.libraries.dialogs_py.MultipleSelectionDialog method), 149
grid_slaves()
(robot.libraries.dialogs_py.PassFailDialog method), 162
gird_slaves()
(robot.libraries.dialogs_py.SelectionDialog method), 136
handle() (robot.output.pyloggingconf.RobotHandler method), 204
handle() (robot.running.arguments.argumentresolver.DictToKwargs method), 277
handle_error() (robot.output.pyloggingconf.RobotHandler method), 204
Handler (in module robot.running.handlers), 290
HandlerExecutionFailed, 337
HandlerStore (class in robot.running.handlerstore), 291
handles() (robot.tree.html.parser.CssFileWriter method), 20
handles() (robot.tree.html.parser.GeneratorWriter method), 20
handles() (robot.tree.html.parser.JsonFileWriter method), 20
handles() (robot.tree.html.parser.LineWriter method), 20
handle_suite_teardown_failures() (robot.result.executionresult.CombinedResult method), 249
handle_suite_teardown_failures() (robot.result.executionresult.Result method), 248
handle_suite_teardown_failures() (robot.result.model.TestSuite method), 264
HandlerError (robot.output.pyloggingconf.RobotHandler method), 204
Handler () (in module robot.running.handlers), 290
HandlerExecutionFailed, 337
HandlerStore (class in robot.running.handlerstore), 291
handles() (robot.tree.html.treefilewriter.CssFileWriter method), 20
handles() (robot.tree.html.treefilewriter.GeneratorWriter method), 20
handle_suite_teardown_failures() (robot.tree.html.treefilewriter.JsonFileWriter method), 20
handle_suite_teardown_failures() (robot.tree.html.treefilewriter.LineWriter method), 20
Handler() (in module robot.running.handlers), 290
HandlerExecutionFailed, 337
HandlerStore (class in robot.running.handlerstore), 291
handles() (robot.htmldata.jsonwriter.IntegerDumper method), 21
handles() (robot.htmldata.jsonwriter.MappingDumper method), 21
handles() (robot.htmldata.jsonwriter.NoneDumper method), 21
handles() (robot.htmldata.jsonwriter.StringDumper method), 21
handles() (robot.htmldata.jsonwriter.TupleListDumper method), 21
handles() (robot.libdocpkg.consoleviewer.ConsoleViewer method), 22
handles() (robot.libdocpkg.htmlwriter.LibdocModelWriter method), 22
handles() (robot.reporting.logreportwriters.RobotModelWriter method), 240
handles() (robot.running.arguments.typeconverters.BooleanConverter method), 279
handles() (robot.running.arguments.typeconverters.ByteArrayConverter method), 280
handles() (robot.running.arguments.typeconverters.ByteConverter method), 280
handles() (robot.running.arguments.typeconverters.DateConverter method), 281
handles() (robot.running.arguments.typeconverters.DateTimeConverter method), 281
handles() (robot.running.arguments.typeconverters.DecimalConverter method), 283
handles() (robot.running.arguments.typeconverters.DictionaryConverter method), 283
handles() (robot.running.arguments.typeconverters.EnumConverter method), 283
handles() (robot.running.arguments.typeconverters.FloatConverter method), 279
handles() (robot.running.arguments.typeconverters.FrozensetConverter method), 284
handles() (robot.running.arguments.typeconverters.IntegerConverter method), 279
handles() (robot.running.arguments.typeconverters.ListConverter method), 282
handles() (robot.running.arguments.typeconverters.NoneConverter method), 282
handles() (robot.running.arguments.typeconverters.SetConverter method), 281
handles() (robot.running.arguments.typeconverters.TimeDeltaConverter method), 281
handles() (robot.running.arguments.typeconverters.TupleConverter method), 283
 handles() (robot.runningsuite.TestSuite attribute), 316
 handles() (robot.result.model.TestSuite attribute), 259
 handles() (robot.result.model.TestSuite attribute), 299
 handles() (robot.running.arguments.typeconverters.DictionaryConverter method), 283
 handles() (robot.running.arguments.typeconverters.KeyedConverter method), 212
 handles() (robot.running.arguments.typeconverters.ListConverter method), 282
 handles() (robot.running.arguments.typeconverters.NoneConverter method), 177
 handles() (robot.running.arguments.typeconverters.SetConverter method), 259
 handles() (robot.running.arguments.typeconverters.TimeDeltaConverter method), 22
 handles() (robot.running.arguments.typeconverters.TupleConverter method), 238
 handles() (robot.running.arguments.typeconverters.TypeConverter method), 320
 handles() (robot.testdoc.TestdocModelWriter method), 345
 handles() (robot.utils.htmlformatters.HeaderFormatter method), 319
 handles() (robot.utils.htmlformatters.LineFormatter method), 318
 handles() (robot.utils.htmlformatters.ListFormatter method), 319
 handles() (robot.utils.htmlformatters.ParagraphFormatter method), 319
 handles() (robot.utils.htmlformatters.PreformattedFormatter method), 319
 handles() (robot.utils.htmlformatters.RulerFormatter method), 319
 handles() (robot.utils.htmlformatters.TableFormatter method), 319
 handles() (robot.utils.importer.ByPathImporter method), 320
 handles() (robot.utils.importer.DottedImporter method), 320
 handles() (robot.utils.importer.NonDottedImporter method), 320
 handles() (robot.utils.restreader.CaptureRobotData attribute), 324
 handles() (robot.utils.restreader.RobotDataStorage method), 324
 handles() (robot.parsing.lexer.tokens.Token attribute), 213
 handles() (robot.parsing.lexer.tokens.EOS attribute), 212
 handles() (robot.output.console.highlighting.HighlightingStream method), 196
 handles() (robot.output.console.highlighting.HighlightingStream method), 196
 handles() (robot.output.console.highlighting.HighlightingStream method), 196
 handles() (robot.output.loggerhelper.Message attribute), 177
 handles() (robot.output.loggerhelper.Message attribute), 202
 handles() (robot.libdocpkg.htmlwriter.DocFormatter method), 316
 handles() (robot.libdocpkg.htmlwriter.LibdocModelWriter method), 22
 handles() (robot.libdocpkg.htmlwriter.TupleListDumper method), 22
 handles() (robot.parsing.lexer.tokens.EOS attribute), 213
 handles() (robot.parsing.lexer.tokens.Token attribute), 212
 handles() (robot.utils.htmlformatters.IntegerDumper method), 319
 handles() (robot.utils.htmlformatters.ListFormatter method), 319
 handles() (robot.utils.htmlformatters.RulerFormatter method), 319
 handles() (robot.utils.htmlformatters.TableFormatter method), 319
 handles() (robot.utils.importer.ByPathImporter method), 320
 handles() (robot.utils.importer.DottedImporter method), 320
 handles() (robot.utils.importer.NonDottedImporter method), 320
 handles() (robot.utils.restreader.CaptureRobotData attribute), 324
 handles() (robot.utils.restreader.RobotDataStorage method), 324
 handles() (robot.model.message.Message attribute), 177
 handles() (robot.model.message.Message attribute), 202
 handles() (robot.libdocpkg.htmlwriter.DocFormatter method), 316
 handles() (robot.libdocpkg.htmlwriter.LibdocModelWriter method), 22
 handles() (robot.reporting.jsbuildingcontext.JsBuildingContext method), 259
 handles() (robot.reporting.jsbuildingcontext.JsBuildingContext method), 196
 handles() (robot.output.console.highlighting.HighlightingStream method), 196
 handles() (robot.output.console.highlighting.HighlightingStream method), 196
 handles() (robot.output.console.highlighting.HighlightingStream method), 196
 handles() (robot.output.loggerhelper.Message attribute), 177
 handles() (robot.output.loggerhelper.Message attribute), 202
 handles() (robot.libdocpkg.htmlwriter.DocFormatter method), 316
 handles() (robot.libdocpkg.htmlwriter.LibdocModelWriter method), 22
 handles() (robot.reporting.jsbuildingcontext.JsBuildingContext method), 259
 handles() (robot.reporting.jsbuildingcontext.JsBuildingContext method), 196
 handles() (robot.output.console.highlighting.HighlightingStream method), 196
 handles() (robot.output.console.highlighting.HighlightingStream method), 196
 handles() (robot.output.console.highlighting.HighlightingStream method), 196
 handles() (robot.output.loggerhelper.Message attribute), 177
 handles() (robot.output.loggerhelper.Message attribute), 202
 handles() (robot.libdocpkg.htmlwriter.DocFormatter method), 316
 handles() (robot.libdocpkg.htmlwriter.LibdocModelWriter method), 22
 handles() (robot.reporting.jsbuildingcontext.JsBuildingContext method), 259
 handles() (robot.reporting.jsbuildingcontext.JsBuildingContext method), 196
 handles() (robot.output.console.highlighting.HighlightingStream method), 196
 handles() (robot.output.console.highlighting.HighlightingStream method), 196
 handles() (robot.output.console.highlighting.HighlightingStream method), 196
 handles() (robot.output.loggerhelper.Message attribute), 177
 handles() (robot.output.loggerhelper.Message attribute), 202
 handles() (robot.libdocpkg.htmlwriter.DocFormatter method), 316
 handles() (robot.libdocpkg.htmlwriter.LibdocModelWriter method), 22
 handles() (robot.reporting.jsbuildingcontext.JsBuildingContext method), 259
 handles() (robot.reporting.jsbuildingcontext.JsBuildingContext method), 196
 handles() (robot.output.console.highlighting.HighlightingStream method), 196
 handles() (robot.output.console.highlighting.HighlightingStream method), 196
 handles() (robot.output.console.highlighting.HighlightingStream method), 196
tribute), 202
html_message (robot.result.model.Message attribute), 259
HtmlFileWriter (class in robot.htmldata.htmlfilewriter), 20
HtmlFormatter (class in robot.util.htmlformatters), 318
HtmlTemplate (class in robot.htmldata.normaltemplat), 21
HtmlWriter (class in robot.util.markupwriters), 321
iconbitmap() (robot.libraries.dialogs.py.InputDialog method), 123
iconbitmap() (robot.libraries.dialogs.py.MessageDialog method), 110
iconbitmap() (robot.libraries.dialogs_py.MultilineSelectionDialog method), 150
iconbitmap() (robot.libraries.dialogs_py.PassFailDialog method), 163
iconbitmap() (robot.libraries.dialogs_py.SelectionDialog method), 137
iconwindow() (robot.libraries.dialogs_py.MessageDialog method), 111
iconwindow() (robot.libraries.dialogs_py.MultilineSelectionDialog method), 150
iconwindow() (robot.libraries.dialogs_py.PassFailDialog method), 163
iconwindow() (robot.libraries.dialogs_py.SelectionDialog method), 137
iconify() (robot.libraries.dialogs_py.InputDialog method), 124
iconify() (robot.libraries.dialogs_py.MessageDialog method), 111
iconify() (robot.libraries.dialogs_py.MultilineSelectionDialog method), 150
iconify() (robot.libraries.dialogs_py.PassFailDialog method), 163
iconify() (robot.libraries.dialogs_py.SelectionDialog method), 137
iconmask() (robot.libraries.dialogs_py.InputDialog method), 124
iconmask() (robot.libraries.dialogs_py.MessageDialog method), 111
iconmask() (robot.libraries.dialogs_py.MultilineSelectionDialog method), 150
iconmask() (robot.libraries.dialogs_py.PassFailDialog method), 163
iconmask() (robot.libraries.dialogs_py.SelectionDialog method), 137
iconname() (robot.libraries.dialogs_py.InputDialog method), 124
iconname() (robot.libraries.dialogs_py.MultilineSelectionDialog method), 150
iconname() (robot.libraries.dialogs_py.PassFailDialog method), 163
iconname() (robot.libraries.dialogs_py.SelectionDialog method), 137
iconposition() (robot.libraries.dialogs_py.MessageDialog method), 111
iconposition() (robot.libraries.dialogs_py.MultilineSelectionDialog method), 150
iconposition() (robot.libraries.dialogs_py.PassFailDialog method), 150
iconposition() (robot.libraries.dialogs_py.SelectionDialog method), 137
iconwindow() (robot.libraries.dialogs_py.MessageDialog method), 111
iconwindow() (robot.libraries.dialogs_py.MultilineSelectionDialog method), 150
iconwindow() (robot.libraries.dialogs_py.PassFailDialog method), 163
iconwindow() (robot.libraries.dialogs_py.SelectionDialog method), 137

identifiers (robot.variables.finders.EmptyFinder attribute), 331
identifiers (robot.variables.finders.EnumeratorFinder attribute), 331
identifiers (robot.variables.finders.ExtendedFinder attribute), 331
identifiers (robot.variables.finders:inlinePythonFinder attribute), 331
identifiers (robot.variables.finders.NumberFinder attribute), 331
identifiers (robot.variables.finders.StoredFinder attribute), 330
IGNORE (robot.parsing.lexer.tokens.EOS attribute), 213
IGNORE (robot.parsing.lexer.tokens.Token attribute), 212
ignored_dirs (robot.parsing.suitestructure.SuiteStructureBuilder attribute), 237
ignored_prefixes (robot.parsing.suitestructure.SuiteStructureBuilder attribute), 237
image (robot.reporting.stringcache.StringIndex attribute), 243
image_names() (robot.libraries.dialogs_py.InputDialog method), 124
image_names() (robot.libraries.dialogs_py.MessageDialog method), 111
image_names() (robot.libraries.dialogs_py.MultilineSelectionDialog method), 124
insert() (robot.model.keyword.Keywords method), 177
insert() (robot.model.message.Messages method), 178
insert() (robot.model.testcase.TestCases method), 188
insert() (robot.running.model.TestSuites method), 191
insert_into_list() (robot.libraries.Collections.Collections method), 52
IntegerConverter (class in robot.running.arguments.typeconverters), 279
IntegerDumper (class in robot.htmldata.jsonwriter), 21
interact() (robot.libraries.Telnet.TelnetConnection method), 93
INTERNAL_UPDATE_FREQUENCY (robot.libraries.Telnet.TelnetConnection attribute), 89
invalidate_import_caches() (in module robot.utils.importer), 320
InvalidForRunner (class in robot.running.steprunner), 306
is_bytes() (in module robot.utils.robottypes2), 327
is_dict_like() (in module robot.utils.robottypes2), 327
is_dict_var() (in module robot.variables.isvar), 331
is_directory (InvalidForRunner), 73
is_dict_variable (in module robot.variables.isvar), 331
is_dict_var() (in module robot.utils.robottypes2), 327
is_pathlike() (in module robot.utils.robottypes2), 327
is_process_running() (robot.libraries.Process.Process method), 73
is_scalar_var() (in module robot.variables.isvar), 331
is_string() (in module robot.utils.robottypes2), 327
is_truthy() (in module robot.utils.robottypes2), 327
is_unicode() (in module robot.utils.robottypes2), 327
is_var() (in module robot.variables.isvar), 331
is_variable (robot.variables.search.VariableMatch attribute), 334
isatty() (in module robot.utils.compat), 314
IsLogged (class in robot.output.loggerhelper), 203
isreadable() (robot.utils.unic.PrettyRepr method), 328
isrecursive() (robot.utils.unic.PrettyRepr method), 329
item_state() (robot.variables.search.VariableSearcher method), 334
ItemList (class in robot.model.itemlist), 175
items() (robot.model.metadata.Metadata method), 179
items() (robot.utils.dotdict.DotDict method), 316
items() (robot.utils.normalizing.NormalizedDict method), 323
items() (robot.variables.evaluation.EvaluationNamespace method), 330
is_directory (robot.parsing.suitestructure.SuiteStructure attribute), 237
is_falsy() (in module robot.utils.robottypes), 327
is_global (robot.running.libraryscopes.GlobalScope attribute), 292
is_global (robot.running.libraryscopes.TestCaseScope attribute), 292
is_number() (in module robot.utils.robottypes2), 327
is_process_running() (robot.libraries.Process.Process method), 73
is_scalar_var() (in module robot.variables.isvar), 331
is_string() (in module robot.utils.robottypes2), 327
is_truthy() (in module robot.utils.robottypes2), 327
is_unicode() (in module robot.utils.robottypes2), 327
is_var() (in module robot.variables.isvar), 331
is_variable (robot.variables.search.VariableMatch attribute), 334
isatty() (in module robot.utils.compat), 314
IsLogged (class in robot.output.loggerhelper), 203
isreadable() (robot.utils.unic.PrettyRepr method), 328
isrecursive() (robot.utils.unic.PrettyRepr method), 329
item_state() (robot.variables.search.VariableSearcher method), 334
ItemList (class in robot.model.itemlist), 175
items() (robot.model.metadata.Metadata method), 179
items() (robot.utils.dotdict.DotDict method), 316
items() (robot.utils.normalizing.NormalizedDict method), 323
items() (robot.variables.evaluation.EvaluationNamespace method), 330
is_directory (robot.parsing.suitestructure.SuiteStructure attribute), 237
is_falsy() (in module robot.utils.robottypes), 327
is_global (robot.running.libraryscopes.GlobalScope attribute), 292
is_number() (in module robot.utils.robottypes2), 327
is_pathlike() (in module robot.utils.robottypes2), 327
is_process_running() (robot.libraries.Process.Process method), 73
is_scalar_var() (in module robot.variables.isvar), 331
is_string() (in module robot.utils.robottypes2), 327
is_truthy() (in module robot.utils.robottypes2), 327
is_unicode() (in module robot.utils.robottypes2), 327
is_var() (in module robot.variables.isvar), 331
is_variable (robot.variables.search.VariableMatch attribute), 334
isatty() (in module robot.utils.compat), 314
IsLogged (class in robot.output.loggerhelper), 203
isreadable() (robot.utils.unic.PrettyRepr method), 328
isrecursive() (robot.utils.unic.PrettyRepr method), 329
item_state() (robot.variables.search.VariableSearcher method), 334
ItemList (class in robot.model.itemlist), 175
items() (robot.model.metadata.Metadata method), 179
items() (robot.utils.dotdict.DotDict method), 316
items() (robot.utils.normalizing.NormalizedDict method), 323
items() (robot.variables.evaluation.EvaluationNamespace method), 330
J
JavaArgumentParser (class in robot.running.arguments.argumentparser), 276
JavaCapturer (class in robot.running.outputcapture), 302
JavaDocBuilder (class in robot.libdocpkg.javabuilder), 23
JavaDocBuilder() (in module robot.libdocpkg.builder), 22
JavaErrorDetails (class in robot.utils.error), 317
join_command_line() (robot.libraries.Process.Process method), 76
join_path() (robot.libraries.OperatingSystem.OperatingSystem method), 67
join_paths() (robot.libraries.OperatingSystem.OperatingSystem method), 68
js_result (robot.reporting.resultwriter.Results attribute), 242
JsBuildingContext (class in robot.reporting.jsbuildingcontext), 238
JsExecutionResult (class in robot.reporting.jsexecutionresult), 239
JsFileWriter (class in robot.htmldata.htmlfilewriter), 20
JsModelBuilder (class in robot.reporting.jsmodelbuilders), 239
JsonConverter (class in robot.libdocpkg.htmlwriter), 22
JsonConverter (class in robot.testdoc), 345
JsonDumper (class in robot.htmldata.jsonwriter), 20
JsonWriter (class in robot.htmldata.jsonwriter), 20
JsResultWriter (class in robot.reporting.jswriter), 240

K
keep_in_dictionary() (robot.libraries.Collections.Collections method), 52
keys() (robot.libraries.dialogs_py.InputDialog method), 124
keys() (robot.libraries.dialogs_py.MessageDialog method), 111
keys() (robot.libraries.dialogs_py.MultilineSelectionDialog method), 150
keys() (robot.libraries.dialogs_py.PassFailDialog method), 163
keys() (robot.libraries.dialogs_py.SelectionDialog method), 137
keys() (robot.model.metadata.Metadata method), 179
keys() (robot.utils.dotdict.DotDict method), 316
keys() (robot.utils.normalizing.NormalizedDict method), 323
keys() (robot.variables.evaluation.EvaluationNamespace method), 330
Keyword (class in robot.model.keyword), 175
Keyword (class in robot.parsing.model.blocks), 216
Keyword (class in robot.result.model), 259
Keyword (class in robot.running.model), 293
KEYWORD (robot.parsing.lexer.tokens.EOS attribute), 213
KEYWORD (robot.parsing.lexer.tokens.Token attribute), 212
Keyword (robot.parsing.model.statements.KeywordCall attribute), 233
keyword() (in module robot.api.deco), 13
keyword_class (robot.model.keyword.Keyword attribute), 175
keyword_class (robot.model.testcase.TestCase attribute), 187
keyword_class (robot.model.testsuite.TestSuite attribute), 189
keyword_class (robot.result.model.Keyword attribute), 261
keyword_class (robot.result.model.TestCase attribute), 261
keyword_class (robot.result.model.TestSuite attribute), 262
keyword_class (robot.running.model.ForLoop attribute), 294
keyword_class (robot.running.model.Keyword attribute), 294
keyword_class (robot.running.model.TestCase attribute), 296
keyword_class (robot.running.model.TestSuite attribute), 297
keyword_context() (robot.parsing.lexer.context.FileContext method), 208
keyword_context() (robot.parsing.lexer.context.InitFileContext method), 209
keyword_context() (robot.parsing.lexer.context.ResourceFileContext method), 208
keyword_context() (robot.parsing.lexer.context.TestCaseContext method), 208
KEYWORD_HEADER (robot.parsing.lexer.tokens.EOS attribute), 213
KEYWORD_HEADER (robot.parsing.lexer.tokens.Token attribute), 211
keyword_marker() (robot.output.console.verbose.VerboseWriter method), 197
KEYWORD_NAME (robot.parsing.lexer.tokens.EOS attribute), 213
KEYWORD_NAME (robot.parsing.lexer.tokens.Token attribute), 213
tribute), 211

keyword_section()
(robot.parsing.lexer.context.FileContext method), 208

keyword_section()
(robot.parsing.lexer.context.InitFileContext method), 209

keyword_section()
(robot.parsing.lexer.context.ResourceFileContext method), 208

keyword_section()
(robot.parsing.lexer.context.TestCaseFileContext method), 208

keyword_should_exist()
(robot.libraries.BuiltIn.BuiltIn method), 33

keyword_timeout
(robot.errors.TimeoutError attribute), 336

KEYWORD_TYPE
(robot.model.keyword.Keyword attribute), 175

KEYWORD_TYPE
(robot.result.model.Keyword attribute), 260

KEYWORD_TYPE
(robot.running.model.ForLoop attribute), 294

KEYWORD_TYPE
(robot.running.model.Keyword attribute), 293

KeywordBuilder
 klass in robot.reporting.jsmodelbuilders, 239

KeywordBuilder
 klass in robot.running.builder.transformers, 288

KeywordCall
 klass in robot.parsing.model.statements, 233

KeywordCallTemplate
 klass in robot.running.arguments.argumentmapper, 276

KeywordContext
 klass in robot.parsing.lexer.context, 209

KeywordDoc
 klass in robot.libdocpkg.model, 23

KeywordDocBuilder
 klass in robot.libdocpkg.robotbuilder, 23

KeywordError, 336

KeywordHandler
 klass in robot.result.xmlelementhandlers, 271

KeywordMarker
 klass in robot.output.console.verbosel, 197

KeywordMatcher
 klass in robot.libdocpkg.consoleviewer, 22

KeywordName
 klass in robot.parsing.model.statements, 229

KeywordRecommendationFinder
 klass in robot.running.namespace, 301

KeywordRemover()
 in robot.result.keywordremover, 250

Keywords
 klass in robot.model.keyword, 176

keywords
 klass in robot.libdocpkg.model.LibraryDoc attribute), 23

keywords
 klass in robot.model.keyword.Keyword attribute), 176

keywords
 klass in robot.model.testcase.TestCase attribute), 188

keywords
 klass in robot.model.testsuite.TestSuite attribute), 189

keywords
 klass in robot.result.model.Keyword attribute), 261

keywords
 klass in robot.result.model.TestCase attribute), 262

keywords
 klass in robot.result.model.TestSuite attribute), 264

keywords
 klass in robot.running.model.ForLoop attribute), 295

keywords
 klass in robot.running.model.Keyword attribute), 294

keywords
 klass in robot.running.model.ResourceFile attribute), 300

keywords
 klass in robot.running.model.TestCase attribute), 296

keywords
 klass in robot.running.model.TestSuite attribute), 299

keywords
 klass in robot.running.model.UserKeyword attribute), 300

KeywordSection
 klass in robot.parsing.model.blocks, 215

KeywordSectionHeader
 klass in robot.parsing.model.statements, 221

KeywordSettings
 klass in robot.parsing.lexer.settings, 211

KeywordStatusHandler
 klass in robot.result.xmlelementhandlers, 272

KeywordStore
 klass in robot.running.namespace, 301

KeywordTimeout
 klass in robot.running.timeouts, 289

KILL_TIMEOUT

kwname
 klass in robot.result.model.Keyword attribute), 259

LastStatementFinder
 klass in robot.parsing.model.blocks, 217

length_should_be()
 klass in robot.libraries.BuiltIn.BuiltIn method), 33

level
 klass in robot.model.message.Message attribute), 177

level
 klass in robot.output.loggerhelper.Message attribute), 202

level
 klass in robot.result.model.Message attribute), 259

lex()
 klass in robot.parsing.lexer.settings.InitFileSettings method), 210

lex()
 klass in robot.parsing.lexer.settings.KeywordSettings method), 211
lines (robot.parsing.model.statements.Comment attribute), 235
lines (robot.parsing.model.statements.CommentSectionHeaders attribute), 222
lines (robot.parsing.model.statements.DefaultTags attribute), 225
lines (robot.parsing.model.statements.Documentation attribute), 224
lines (robot.parsing.model.statements.DocumentationOrMetadata attribute), 218
lines (robot.parsing.model.statements.EmptyLine attribute), 236
lines (robot.parsing.model.statements.End attribute), 235
lines (robot.parsing.model.statements.Error attribute), 236
lines (robot.parsing.model.statements.Fixture attribute), 219
lines (robot.parsing.model.statements.ForceTags attribute), 225
lines (robot.parsing.model.statements.ForLoopHeader attribute), 235
lines (robot.parsing.model.statements.KeywordCall attribute), 234
lines (robot.parsing.model.statements.KeywordName attribute), 230
lines (robot.parsing.model.statements.KeywordSectionHeader attribute), 222
lines (robot.parsing.model.statements.LibraryImport attribute), 223
lines (robot.parsing.model.statements.Metadata attribute), 225
lines (robot.parsing.model.statements.MultiValue attribute), 219
lines (robot.parsing.model.statements.ResourceImport attribute), 223
lines (robot.parsing.model.statements.Return attribute), 233
lines (robot.parsing.model.statements.SectionHeader attribute), 220
lines (robot.parsing.model.statements.SettingSectionHeader attribute), 230
lines (robot.parsing.model.statements.SingleValue attribute), 218
lines (robot.parsing.model.statements.Statement attribute), 218
lines (robot.parsing.model.statements.SuiteSetup attribute), 226
lines (robot.parsing.model.statements.SuiteTeardown attribute), 226
lines (robot.parsing.model.statements.Tags attribute), 231
lines (robot.parsing.model.statements.Teardown attribute), 231
lines (robot.parsing.model.statements.Template attribute), 232
lines (robot.parsing.model.statements.TemplateArguments attribute), 234
lines (robot.parsing.model.statements.TestCaseName attribute), 229
lines (robot.parsing.model.statements.TestCaseSectionHeader attribute), 221
lines (robot.parsing.model.statements.TestSetup attribute), 227
lines (robot.parsing.model.statements.TestTeardown attribute), 227
lines (robot.parsing.model.statements.TestTemplate attribute), 228
lines (robot.parsing.model.statements.Timeout attribute), 232
lines (robot.parsing.model.statements.Variable attribute), 229
lines (robot.parsing.model.statements.VariableSectionHeader attribute), 221
lines (robot.parsing.model.statements.VariablesImport attribute), 224
lines (robot.parsing.model.statements.Writer (class in robot.htmldata.htmlfilewriter), 20
lines (robot.parsing.model.statements.Library Import attribute), 223
lines (robot.parsing.model.statements.LibraryImport attribute), 223
link () (robot.reporting.jsbuildingcontext.JsBuildingContext method), 238
LinkFormatter (class in robot.utils.htmlformatters, 318
links (robot.model.stats.TagStat attribute), 183
list () (robot.libdocpkg.consoleviewer.ConsoleViewer method), 22
list_directories_in_directory ()

list_directory () (robot.libraries.OperatingSystem.OperatingSystem method), 69
list_files_in_directory () (robot.libraries.OperatingSystem.OperatingSystem method), 70
list_should_contain_sub_list () (robot.libraries.Collections.Collections method), 52
list_should_contain_value () (robot.libraries.Collections.Collections method), 53
list_should_not_contain_duplicates () (robot.libraries.Collections.Collections method), 53
list_should_not_contain_value () (robot.libraries.Collections.Collections method), 53
message (robot.errors.ReturnFromKeyword attribute), 340
message (robot.errors.RobotError attribute), 335
message (robot.errors.TimeoutError attribute), 336
message (robot.errors.UserKeywordExecutionFailed attribute), 338
message (robot.errors.VariableError attribute), 336
message (robot.libraries.BuiltIn.RobotNotRunningError attribute), 47
message (robot.libraries.Telnet.NoMatchError attribute), 94
message (robot.model.message.Message attribute), 177
message (robot.model.totalstatistics.TotalStatistics attribute), 191
message (robot.output.loggerhelper.MessageStatistics attribute), 305
message (robot.output.status.SuiteMessage attribute), 305
message (robot.output.status.SuiteStatus attribute), 304
message (robot.output.status.TestMessage attribute), 305
message (robot.output.status.TestStatus attribute), 305
message (robot.utils.error.JavaErrorDetails attribute), 317
message (robot.utils.error.HtmlErrorDetails attribute), 317
message () (robot.output.console.dotted.DottedOutput method), 195
message () (robot.output.console.quiet.QuietOutput method), 197
message () (robot.output.console.verbosity.VerboseOutput method), 197
message () (robot.output.console.verbosity.VerboseWriter method), 197
message () (robot.output.filelogger.FileLogger method), 198
message () (robot.output.logger.Logger method), 201
message () (robot.output.loggerhelper.AbstractLogger method), 202
message () (robot.output.output.Output method), 203
message () (robot.output.xmllogger.XmlLogger method), 205
message () (robot.reporting.outputwriter.OutputWriter method), 241
message_class (robot.model.keyword.Keyword attribute), 175
message_class (robot.result.executionerrors.ExecutionErrors attribute), 247
message_class (robot.result.model.Keyword attribute), 257
message_class (robot.running.model.ForLoop attribute), 259
message_class (robot.running.model.Keyword attribute), 293
message_class (robot.running.model.TestSuite attribute), 295
message_level() (robot.reporting.jsbuildingcontext.JsBuildingContext method), 238
MessageArguments (class in robot.output.listenerarguments), 199
MessageBuilder (class in robot.reporting.jsmodelbuilders), 239
MessageDialog (class in robot.libraries.dialogs_py), 106
MessageFilter (class in robot.result.messagefilter), 257
MessageHandler (class in robot.result.messagefilter), 272
Messages (class in robot.model.message), 178
messages (robot.model.keyword.Keyword attribute), 176
messages (robot.result.executionerrors.ExecutionErrors attribute), 247
messages (robot.result.model.Keyword attribute), 261
messages (robot.running.model.ForLoop attribute), 295
messages (robot.running.model.Keyword attribute), 294
Metadata (class in robot.model.metadata), 179
Metadata (class in robot.parsing.model.statements), 224
metadata (robot.model.testsuite.TestSuite attribute), 189
METADATA (robot.parsing.lexer.tokens.EOS attribute), 213
METADATA (robot.parsing.lexer.tokens.Token attribute), 211
metadata (robot.result.model.TestSuite attribute), 265
metadata (robot.running.model.Testsuite attribute), 299
MetadataItemHandler (class in robot.result.xmlelementhandlers), 272
MetadataItemHandler (class in robot.result.xmlelementhandlers), 273
minargs (robot.running.arguments.argumentspec.ArgumentSpec attribute), 277
minsize() (robot.libraries.dialogs_py.InputDialog method), 124
minsize() (robot.libraries.dialogs_py.MessageDialog method), 111
minsize() (robot.libraries.dialogs_py.MessageDialog method), 150
minsize() (robot.libraries.dialogs_py.PassFailDialog method), 163
minsize() (robot.libraries.dialogs_py.SelectionDialog method), 137
mode_and_args() (robot.tidy.ArgumentValidator method), 347
ModelCombiner (class in robot.running.runner), 304
ModelModifier (class in robot.model.modifier), 180
ModelObject (class in robot.model.modelobject), 179
ModelTransformer (class in robot.parsing.model.visitor), 237
ModelVisitor (class in robot.parsing.model.visitor), 237
ModelWriter (class in robot.parsing.model.visitor), 237
ModelWriter (class in robot.htmldata.htmlfilewriter), 20
move_directory() (robot.libraries.Telnet.TelnetConnection method), 67
move_file() (robot.libraries.Telnet.TelnetConnection method), 66
move_files() (robot.libraries.Telnet.TelnetConnection method), 66
mro() (robot.utils.setter.SetterAwareType method), 328
msg() (robot.libraries.Telnet.TelnetConnection method), 92
mt_interact() (robot.libraries.Telnet.TelnetConnection method), 93
multi_use (robot.parsing.lexer.settings.InitFileSettings attribute), 210
multi_use (robot.parsing.lexer.settings.KeywordSettings attribute), 211
multi_use (robot.parsing.lexer.settings.ResourceFileSettings attribute), 210
multi_use (robot.parsing.lexer.settings.Settings attribute), 209
multi_use (robot.parsing.lexer.settings.TestCaseSettings attribute), 210
multi_use (robot.parsing.lexer.settings.TestCaseSettings attribute), 211
MultiMatcher (class in robot.utils.match), 322
MultipleSelectionDialog (class in robot.libraries.dialogs_py), 145
MultiValue (class in robot.parsing.model.statements), 218

NAME (robot.parsing.lexer.tokens.Token attribute), 212
name (robot.parsing.model.blocks.Keyword attribute), 216
name (robot.parsing.model.blocks.TestCase attribute), 216
name (robot.parsing.model.statements.Fixture attribute), 219
name (robot.parsing.model.statements.KeywordName attribute), 229
name (robot.parsing.model.statements.LibraryImport attribute), 222
name (robot.parsing.model.statements.Metadata attribute), 224
name (robot.parsing.model.statements.ResourceImport attribute), 223
name (robot.parsing.model.statements.Setup attribute), 230
name (robot.parsing.model.statements.SuiteSetup attribute), 226
name (robot.parsing.model.statements.SuiteTeardown attribute), 226
name (robot.parsing.model.statements.Teardown attribute), 231
name (robot.parsing.model.statements_TestCaseName attribute), 229
name (robot.parsing.model.statements_TestCaseSetup attribute), 227
name (robot.parsing.model.statements_TestCaseTeardown attribute), 227
name (robot.parsing.model.statements_Variable attribute), 228
name (robot.parsing.model.statements_VariablesImport attribute), 223
name (robot.result.model.Keyword attribute), 260
name (robot.result.model_TestCase attribute), 262
name (robot.result.model_TestCaseSuite attribute), 265
name (robot.running.dynamicmethods_GetKeywordArguments attribute), 290
name (robot.running.dynamicmethods_GetKeywordDocumentation attribute), 290
name (robot.running.dynamicmethods_GetKeywordNames attribute), 290
name (robot.running.dynamicmethods_GetKeywordTags attribute), 290
name (robot.running.dynamicmethods_RunKeyword attribute), 290
name (robot.running.model.ForLoop attribute), 295
name (robot.running.model.Keyword attribute), 294
name (robot.running.model_TestCase attribute), 296
name (robot.running.model_TestCaseSuite attribute), 299
name (robot.variables.search.VariableMatch attribute), 333

Index
output_file() (robot.output.console.implicit.VerboseOutput method), 197
output_file() (robot.output.filelogger.FileLogger method), 198
output_file() (robot.output.listeners.Listeners method), 200
output_file() (robot.output.listeners.Listeners method), 200
output_file() (robot.output.logger.Logger method), 201
OutputCapturer (class in robot.running.outputcapture), 302
OutputWriter (class in robot.reporting.outputwriter), 302
overrideredirect()
(robot.libraries.dialogs_py.InputDialog method), 125
overrideredirect()
(robot.libraries.dialogs_py.MessageDialog method), 112
overrideredirect()
(robot.libraries.dialogs_py.MultilineSelectionDialog method), 151
overrideredirect()
(robot.libraries.dialogs_py.PassFailDialog method), 164
overrideredirect()
(robot.libraries.dialogs_py.SelectionDialog method), 138
pack_propagate()
pack_propagate()
(robot.libraries.dialogs_py.MessageDialog arguments() (robot.libdoc.LibDoc method), 342
pack_propagate()
(robot.libraries.dialogs_py.MultilineSelectionDialog arguments() (robot.run.RobotFramework method), 343
pack_propagate()
pack_propagate()
(robot.libraries.dialogs_py.SelectionDialog arguments() (robot.run.RobotFramework method), 347
pack_slaves()
(robot.libraries.dialogs_py.InputDialog parse_args() (robot.rebot.Rebot method), 125
pack_slaves()
pack_slaves()
(robot.libraries.dialogs_py.MultilineSelectionDialog parse_args() (robot.tidy.TidyCommandLine method), 151
pack_slaves()
(robot.libraries.dialogs_py.PassFailDialog parse_args() (robot.utils.application.Application method), 164
pack_slaves()
(robot.libraries.dialogs_py.SelectionDialog parse_args() (robot.utils.application.Application method), 138
pad_console_length()
(robot.utils.text), 328
(robot.running.builder.parsers.RobotParser method), 285
parse_resource_file() (robot.running.builder.parsers.BaseParser method), 285
parse_resource_file() (robot.running.builder.parsers.NoInitDirectoryParser method), 285
parse_resource_file() (robot.running.builder.parsers.RestParser method), 285
parse_resource_file() (robot.running.builder.parsers.RobotParser method), 285
parse_response() (robot.libraries.Remote.TimeoutHTTPTransport method), 78
parse_response() (robot.libraries.Remote.TimeoutHTTPTransport method), 77
parse_suite_file() (robot.running.builder.parsers.BaseParser method), 285
parse_suite_file() (robot.running.builder.parsers.NoInitDirectoryParser method), 285
parse_suite_file() (robot.running.builder.parsers.RestParser method), 285
parse_time() (in module robot.utilities), 326
parse_xml() (robot.libraries.XML.XML method), 326
pass_execution() (robot.libraries.BuiltIn.Builtin method), 99
pass_execution_if() (robot.libraries.BuiltIn.Builtin method), 34
passed (robot.model.itemlist.ItemList method), 316
passed (robot.model.keyword.Keywords method), 182
passed (robot.model.testcase.TestCases method), 260
passed (robot.model.testsuite.TestSuites method), 261
PassedKeywordRemover (class in robot.libraries.BuiltIn), 251
PassExecution, 338
PassFailDialog (class in robot.libraries.BuiltIn), 158
path_to_url() (in module robot.utils.path), 158
pause_execution() (in module robot.libraries.Debugee), 60
pformat() (robot.utils.htmlformatters.PrettyRepr method), 329
place_slaves() (robot.libraries.dials.dialogs_py.MultipleSelectionDialog method), 151
place_slaves() (robot.libraries.dials.dialogs_py.PassFailDialog method), 164
place_slaves() (robot.libraries.dials.dialogs_py.SelectionDialog method), 138
pre_rebot_modifiers (robot.conf.settings.RebotSettings attribute), 18
pre_rebot_modifiers (robot.conf.settings.RebotSettings attribute), 19
pre_run_modifiers (robot.conf.settings.RebotSettings attribute), 18
PreformattedFormatter (class in robot.libraries.htmlformatters.formatters), 319
pretty_print() (robot.utils.htmlformatters.PrettyRepr method), 329
printable_name() (in module robot.utils.misc), 322
Process (class in robot.libraries.Process), 70

Index 409
process() (robot.utils.argumentparser.ArgFileParser method), 311
process_empty_suite (robot.conf.settings.RebotSettings attribute), 19
process_rawq() (robot.libraries.Telnet.TelnetConnection method), 93
process_should_be_running() (robot.libraries.Process.Process method), 74
process_should_be_stopped() (robot.libraries.Process.Process method), 74
ProcessConfiguration (class in robot.libraries.Process), 76
propagate() (robot.libraries.dialogs_py.InputDialog method), 125
propagate() (robot.libraries.dialogs_py.MessageDialog method), 112
propagate() (robot.libraries.dialogs_py.MultilineSelectionDialog method), 151
propagate() (robot.libraries.dialogs_py.PassFailDialog method), 164
propagate() (robot.libraries.dialogs_py.SelectionDialog method), 138
protocol() (robot.libraries.dialogs_py.InputDialog method), 125
protocol() (robot.libraries.dialogs_py.MessageDialog method), 112
protocol() (robot.libraries.dialogs_py.MultilineSelectionDialog method), 151
protocol() (robot.libraries.dialogs_py.PassFailDialog method), 164
protocol() (robot.libraries.dialogs_py.SelectionDialog method), 138
prune_input() (robot.reporting.jsbuildingcontext.JsBuildingContext method), 238
py2to3() (in module robot.libraries.compat), 314
PythonArgumentParser (class in robot.running.arguments.argumentparser), 276
PythonCapturer (class in robot.running.outputcapture), 302
PythonErrorDetails (class in robot.libraries.error), 317
PythonImporter (class in robot.variables.filesetter), 330
quit() (robot.libraries.dialogs_py.MultilineSelectionDialog method), 151
quit() (robot.libraries.dialogs_py.PassFailDialog method), 164
quit() (robot.libraries.dialogs_py.SelectionDialog method), 138
raise_error() (robot.utils.connectioncache.NoConnection method), 315
randomize() (robot.running.model.TestSuite method), 297
randomize_seed (robot.conf.settings.RobotSettings attribute), 18
randomize_suites (robot.conf.settings.RobotSettings attribute), 18
randomize_tests (robot.conf.settings.RobotSettings attribute), 18
randomizer (class in robot.running.randomizer), 302
rawq_getchar() (robot.libraries.Telnet.TelnetConnection method), 93
read() (robot.libraries.Telnet.TelnetConnection method), 92
read() (robot.libraries.Telnet.TerminalEmulator method), 94
read() (robot.utils.filereader.FileReader method), 318
read_all() (robot.libraries.Telnet.TelnetConnection method), 93
read_eager() (robot.libraries.Telnet.TelnetConnection method), 93
read_lazy() (robot.libraries.Telnet.TelnetConnection method), 93
read_rest_data() (in module robot.utils), 310
read_rest_data() (in module robot.utils.restreader), 324
read_sb_data() (robot.libraries.Telnet.TelnetConnection method), 94
read_some() (robot.libraries.Telnet.TelnetConnection method), 94
read_until() (robot.libraries.Telnet.TelnetConnection method), 92
read_until() (robot.libraries.Telnet.TerminalEmulator method), 94
read_until_prompt() (robot.libraries.Telnet.TelnetConnection method), 92
read_until_regexp() (robot.libraries.Telnet.TelnetConnection method), 92
read_until_regexp() (robot.libraries.Telnet.TerminalEmulator method), 94
read_very_eager() (robot.libraries.Telnet.TelnetConnection method), 94

Q
QuietOutput (class in robot.output.console.quiet), 197
quit() (robot.libraries.dialogs_py.InputDialog method), 125
quit() (robot.libraries.dialogs_py.MessageDialog method), 112
method), 94
read_every_lazy() (robot.libraries.Telnet.TelnetConnection method), 94
readlines() (robot.utils.filereader.FileReader method), 318
real (robot.reporting.stringcache.StringIndex attribute), 243
Rebot (class in robot.rebot), 342
rebot() (in module robot), 11
rebot() (in module robot.rebot), 342
rebot_cli() (in module robot), 11
rebot_cli() (in module robot.rebot), 342
RebotSettings (class in robot.conf.settings), 19
recommend_similar_keywords() (robot.running.namespace.KeywordRecommendationFinder () (robot.parsing.model.statements.KeywordSectionHeader class method), 200
RecommendationFinder (class in robot.utils.recommendations), 323
red() (robot.output.console.highlighting.AnsiHighlighter method), 196
red() (robot.output.console.highlighting.DosHighlighter method), 196
red() (robot.output.console.highlighting.NoHighlighting method), 196
regexp_escape() (robot.libraries.Builtin.Builtin method), 35
register() (robot.libraries.dialogs_py.InputDialog method), 125
register() (robot.libraries.dialogs_py.MessageDialog method), 112
register() (robot.libraries.dialogs_py.SelectionDialog method), 221
register() (robot.libraries.dialogs_py.MultipleSelectionDialog method), 151
register() (robot.libraries.dialogs_py.PassFailDialog method), 164
register() (robot.libraries.dialogs_py.SelectionDialog method), 138
register() (robot.output.listenermethods.LibraryListener method), 200
register() (robot.output.listeners.LibraryListeners method), 200
register() (robot.parsing.model.statements.Arguments class method), 233
register() (robot.parsing.model.statements.Comment class method), 235
register() (robot.parsing.model.statements.CommentSectionHeader class method), 222
register() (robot.parsing.model.statements.DefaultTags class method), 222
register() (robot.parsing.model.statements.DefaultTags class method), 225
register() (robot.parsing.model.statements.DocumentationOrMetadata class method), 224
register() (robot.parsing.model.statements.DocumentationOrMetadata class method), 218
register() (robot.parsing.model.statements.EmptyLine class method), 236
register() (robot.parsing.model.statements.End class method), 235
register() (robot.parsing.model.statements.Error class method), 236
register() (robot.parsing.model.statements.Fixture class method), 219
register() (robot.parsing.model.statements.ForceTags class method), 225
register() (robot.parsing.model.statements.ForLoopHeader class method), 235
register() (robot.parsing.model.statements.KeywordCall class method), 234
register() (robot.parsing.model.statements.KeywordName class method), 230
register() (robot.parsing.model.statements.LibraryImport class method), 222
register() (robot.parsing.model.statements.LibraryImport class method), 223
register() (robot.parsing.model.statements.LibraryImport class method), 225
register() (robot.parsing.model.statements.LibraryImport class method), 219
register() (robot.parsing.model.statements.Metadata class method), 223
register() (robot.parsing.model.statements.Return class method), 233
register() (robot.parsing.model.statements.SectionHeader class method), 220
register() (robot.parsing.model.statements.SettingSectionHeader class method), 220
register() (robot.parsing.model.statements.Setup class method), 230
register() (robot.parsing.model.statements.TestDiscovery class method), 226
register() (robot.parsing.model.statements.TestDiscovery class method), 226
register() (robot.parsing.model.statements.Teardown class method), 231
register() (robot.parsing.model.statements.Teardown class method), 233
register() (robot.parsing.model.statements.Template class method), 231
register() (robot.parsing.model.statements.Template class method), 232
register() (robot.parsing.model.statements.Template class method), 234
register() (robot.parsing.model.statements_TestCaseName class method), 229
register() (robot.parsing.model.statements_TestCaseName class method), 221
register() (robot.parsing.model.statements_TestCaseName class method), 221
register() (robot.parsing.model.statements_TestCaseSectionHeader class method), 227
register() (robot.parsing.model.statements.TestTeardown class method), 227
register_error_listener() (robot.output.output.Logger method), 201
register() (robot.parsing.model.statements.TestTemplate class method), 228
register_listeners() (robot.output.logger.Logger method), 201
register() (robot.parsing.model.statements.TestTimeout class method), 228
register_logger() (robot.output.logger.Logger method), 201
register() (robot.parsing.model.statements.Timeout class method), 232
register_run_keyword() (in module robot.libraries.BuiltIn), 47
register() (robot.parsing.model.statements.Variable class method), 229
register_syslog() (robot.output.logger.Logger method), 201
register() (robot.parsing.model.statements.VariableSectionHeader class method), 221
register() (robot.parsing.model.statements.VariablesImport class method), 224
register() (robot.running.arguments.typeconverters.BooleanConverter class method), 238
release() (robot.output.pylongingconf.RobotHandler class method), 238
register() (robot.running.arguments.typeconverters.ByteArrayConverter class method), 280
release() (robot.running.outputcapture.JavaCapturer class method), 280
register() (robot.running.arguments.typeconverters.ByteStringConverter class method), 280
release() (robot.running.outputcapture.PythonCapturer class method), 280
register() (robot.running.arguments.typeconverters.DateConverter class method), 302
reload_library() (robot.libraries.BuiltIn.BuiltIn class method), 35
register() (robot.running.arguments.typeconverters.DateTimeConverter class method), 281
reload_library() (robot.running.namespace.Namespace class method), 301
register() (robot.running.arguments.typeconverters.DecimalConverter class method), 280
release() (robot.running.outputcapture.PythonCapturer class method), 302
register() (robot.running.arguments.typeconverters.DateTimeConverter class method), 283
release() (robot.running.outputcapture.JavaCapturer class method), 283
register() (robot.running.arguments.typeconverters.DateTimeDeltaConverter class method), 282
release() (robot.running.outputcapture.PythonCapturer class method), 265
register() (robot.running.arguments.typeconverters.DateTimeDeltaConverter class method), 282
release() (robot.running.outputcapture.JavaCapturer class method), 175
register() (robot.running.arguments.typeconverters.FrozenSetConverter class method), 284
release() (robot.running.outputcapture.JavaCapturer class method), 284
register() (robot.running.arguments.typeconverters.IntegerConverter class method), 279
release() (robot.running.outputcapture.JavaCapturer class method), 185
register() (robot.running.arguments.typeconverters.ListConverter class method), 282
remove() (robot.running.arguments.typeconverters.ListConverter class method), 188
register() (robot.running.arguments.typeconverters.NoneConverter class method), 282
remove() (robot.running.arguments.typeconverters.ListConverter class method), 191
register() (robot.running.arguments.typeconverters.SetConverter class method), 283
remove() (robot.running.output.capture.JsExecutionResult.JsExecutionResult class method), 300
register() (robot.running.arguments.typeconverters.TimeDeltaConverter class method), 281
remove_data_not_needed_in_report() (robot.variables.store.VariableStore class method), 334
register() (robot.running.arguments.typeconverters.TupleConverter class method), 283
remove() (robot.reporting.jsexecutionresult.JsExecutionResult class method), 239
register() (robot.running.arguments.typeconverters.TupleConverter class method), 278
remove_directory() (robot.libraries.OperatingSystem.OperatingSystem class method), 66
register() (robot包包.connectioncache.ConnectionCache class method), 315
remove_duplicates() (robot.libraries.Collections.Collections class method), 54
register_console_logger() (robot.output.logger.Logger method), 201
remove_element() (robot.libraries.XML class method), 104
register_error_listener() (robot.output.logger.Logger method), 201
register_logger() (robot.output.logger.Logger method), 201
register() (robot.output.output.Output method), 203
register() (robot.parsing.model.statements.VariablesImport class method), 224
register() (robot.running.arguments.typeconverters.BooleanConverter class method), 238
release() (robot.output.pylongingconf.RobotHandler class method), 238
register() (robot.running.arguments.typeconverters.ByteArrayConverter class method), 280
release() (robot.running.outputcapture.JavaCapturer class method), 280
register() (robot.running.arguments.typeconverters.ByteStringConverter class method), 280
release() (robot.running.outputcapture.PythonCapturer class method), 280
register() (robot.running.arguments.typeconverters.DateConverter class method), 302
reload_library() (robot.libraries.BuiltIn.BuiltIn class method), 35
register() (robot.running.arguments.typeconverters.DateTimeConverter class method), 281
reload_library() (robot.running.namespace.Namespace class method), 301
register() (robot.running.arguments.typeconverters.DecimalConverter class method), 280
release() (robot.running.outputcapture.PythonCapturer class method), 302
register() (robot.running.arguments.typeconverters.DateTimeConverter class method), 283
release() (robot.running.outputcapture.JavaCapturer class method), 283
register() (robot.running.arguments.typeconverters.DateTimeDeltaConverter class method), 282
release() (robot.running.outputcapture.PythonCapturer class method), 265
register() (robot.running.arguments.typeconverters.DateTimeDeltaConverter class method), 282
release() (robot.running.outputcapture.JavaCapturer class method), 175
register() (robot.running.arguments.typeconverters.DateTimeDeltaConverter class method), 282
release() (robot.running.outputcapture.JavaCapturer class method), 185
register() (robot.running.arguments.typeconverters.ListConverter class method), 282
remove() (robot.running.arguments.typeconverters.ListConverter class method), 191
register() (robot.running.arguments.typeconverters.SetConverter class method), 283
remove() (robot.running.output.capture.JsExecutionResult.JsExecutionResult class method), 300
register() (robot.running.arguments.typeconverters.SetConverter class method), 283
remove_data_not_needed_in_report() (robot.variables.store.VariableStore class method), 334
register() (robot.running.arguments.typeconverters.TupleConverter class method), 283
remove() (robot.reporting.jsexecutionresult.JsExecutionResult class method), 239
register() (robot.running.arguments.typeconverters.TupleConverter class method), 278
remove_directory() (robot.libraries.OperatingSystem.OperatingSystem class method), 66
register() (robot包包.connectioncache.ConnectionCache class method), 315
remove_duplicates() (robot.libraries.Collections.Collections class method), 54
register_console_logger() (robot.output.logger.Logger method), 201
remove_element() (robot.libraries.XML class method), 104
register_error_listener() (robot.output.logger.Logger method), 201
register_logger() (robot.output.logger.Logger method), 201
register() (robot.output.output.Output method), 203
report_error() (robot.variables.tablesetter.ListVariableTableValue method), 335
report_error() (robot.variables.tablesetter.ScalarVariableTableValue method), 334
report_invalid_syntax() (robot.running.model.Import method), 300
report_invalid_syntax() (robot.running.model.Variable method), 300
ReportWriter (class in robot.reporting.logreportwriters), 240
reset() (robot.output.console.highlighting.AnsiHighlighter method), 196
reset() (robot.output.console.highlighting.DosHighlighter method), 196
reset() (robot.output.console.highlighting.NoHighlighting method), 196
reset_count() (robot.output.console verbose.KeywordMarker method), 197
resolve() (robot.variables.assigner.ScalarsAndListReturnValueResolver method), 329
resolve() (robot.variables.assigner.ScalarsOnlyReturnValueResolver method), 329
resolve() (robot.variables.tablesetter.DictVariableTableValue method), 335
resolve() (robot.variables.tablesetter.ListVariableTableValue method), 335
resolve() (robot.variables.tablesetter.ScalarVariableTableValue method), 334
resolve() (robot.variables.tablesetter.VariableTableValueBase method), 334
resolve_alias_or_index() (robot.utils.connectioncache.ConnectionCache method), 315
resolve_base() (robot.variables.search.VariableMatch method), 333
resolve_delayed() (robot.variables.scopes.GlobalVariables method), 333
resolve_delayed() (robot.variables.scopes.VariableScopes method), 333
resolve_delayed() (robot.variables.store.VariableStore method), 334
resolve_delayed() (robot.variables.variables.Variables method), 335
resolve_delayed_message() (robot.output.loggerhelper.Message method), 202
resizable() (robot.librariesdialogs_py.InputDialog method), 125
resizable() (robot.librariesdialogs_py.MessageDialog method), 112
resizable() (robot.librariesdialogs_py_MULTIPLE_SELECTION_DIALOG method), 151
resizable() (robot.librariesdialogs_py.PassFailDialog method), 164
resizable() (robot.librariesdialogs_py.SelectionDialog method), 138
resolve() (robot.running.arguments.argumentmapper.DefaultValue method), 276
resolve() (robot.running.arguments.argumentresolver.ArgumentResolver method), 277
resolve() (robot.running.arguments.argumentresolver.NamedArgumentResolver method), 277
resolve() (robot.running.arguments.argumentspec.ArgumentSpec method), 277
resolve() (robot.running.arguments.argumentspec.NamedArgumentResolver method), 277
resolve() (robot.running.arguments.argumentspec.NoReturnValueResolver method), 329
resolve() (robot.running.arguments.argumentspec.ONE_RETURN_VALUE_RESOLVER method), 329
resolve() (robot.running.arguments.argumentspec.RESOURCE_FILE_TYPE method), 301
resolve() (robot.running.arguments.argumentspec.USER_LIBRARY_ATTRIBUTE method), 307
resolve() (robot.running.arguments.argumentspec.USER_LIBRARY_ATTRIBUTE method), 307
resolve() (robot.running.handlerstore.HandlerStore attribute), 287
resolve() (robot.running.model.Imports method), 301
resolve() (robot.running.model.Imports method), 301
resolve() (robot.running.model.TestSuite attribute), 297
resolve() (robot.running.model.UseKeyword method), 297
ResourceBuilder (class in robot.running.builder.transformers), 287
ResourceDocBuilder (class in robot.libdocpkg.robotbuilder), 23
ResourceFile (class in robot.running.model), 300
ResourceFileBuilder (class in robot.running.builder.builders), 285
ResourceFileContext (class in robot.parsing.lexer.context), 208
ResourceFileSettings (class in robot.parsing.lexer.settings), 210
ResourceImport (class in robot.parsing.model.statements), 223
RestParser (class in robot.running.builder.parsers), 285
Result (class in robot.result.executionresult), 247
result_config (robot.libraries.process.ProcessConfiguration attribute), 76
Results (class in robot.reporting.resultwriter), 242
ResultVisitor (class in robot.result.visitor), 269
ResultWriter (class in robot.reporting.resultwriter), 242
Return (class in robot.parsing.model.statements), 233
RETURN (robot.parsing.lexer.tokens.EOS attribute), 213
RETURN (robot.parsing.lexer.tokens.Token attribute), 212
return_code (robot.result.executionresult.Result at-tribute), 249
return_code (robot.result.executionresult.Result attribute), 248
return_from_keyword() (robot.libraries.BuiltIn.BuiltIn method), 36
return_from_keyword_if () (robot.libraries.BuiltIn.BuiltIn method), 36
ReturnFromKeyword, 339
ReturnValueResolver () (in module robot.variables.assigner), 329
reverse() (robot.model.itemlist.ItemList method), 175
reverse() (robot.model.keyword.Keywords method), 177
reverse() (robot.model.message.Messages method), 178
reverse() (robot.model.testcase.TestCases method), 188
reverse() (robot.model.testsuite.TestSuites method), 191
reverse() (robot.running.model.Imports method), 301
reverse_list () (robot.libraries.Collections.Collections method), 54
robot (module), 9
robot.api (module), 7, 12
robot.api.deco (module), 12
robot.api.logger (module), 14
robot.conf (module), 15
robot.conf.gatherfailed (module), 16
robot.conf.settings (module), 18
robot.errors (module), 335
robot.htmldata (module), 20
robot.htmldata.htmlfilewriter (module), 20
robot.htmldata.jsonwriter (module), 20
robot.htmldata.normaltemplate (module), 21
robot.htmldata.template (module), 21
robot.libdoc (module), 340
robot.libdocpkg (module), 21
robot.libdocpkg.builder (module), 22
robot.libdocpkg.consoleviewer (module), 22
robot.libdocpkg.htmlwriter (module), 22
robot.libdocpkg.javabuilder (module), 23
robot.libdocpkg.model (module), 23
robot.libdocpkg.output (module), 23
robot.libdocpkg.robotbuilder (module), 23
robot.libdocpkg.specbuilder (module), 24
robot.libdocpkg.writer (module), 24
robot.libdocpkg.xmlwriter (module), 24
robot.libraries (module), 24
robot.libraries.Builtin (module), 24
robot.libraries.Collections (module), 48
robot.libraries.DateTime (module), 54
robot.libraries.Dialogs (module), 59
robot.libraries.dialogs_py (module), 106
robot.libraries.Easter (module), 60
robot.libraries.Operating System (module), 61
robot.libraries.Process (module), 70
robot.libraries.Remote (module), 76
robot.libraries.Reserved (module), 78
robot.libraries.Screenshot (module), 78
robot.libraries.String (module), 80
robot.libraries.Telnet (module), 85
robot.libraries.XML (module), 95
robot.model (module), 171
robot.model.configurer (module), 171
robot.model.criticality (module), 172
robot.model.filter (module), 172
robot.model.itemlist (module), 175
robot.model.keyword (module), 175
robot.model.message (module), 177
robot.model.metadata (module), 179
robot.model.modelobject (module), 179
robot.model.modifier (module), 180
robot.model.namepatterns (module), 181
robot.libraries.Operating System (module), 61
robot.model.stats (module), 182
robot.model.suitestatistics (module), 184
robot.model.tags (module), 185
robot.model.tagsetter (module), 185
robot.model.tagstatistics (module), 186
robot.model.testcase (module), 187
robot.model.testsuite (module), 189
robot.model.totalstatistics (module), 191
robot.model.visitor (module), 192
robot.output (module), 194
robot.output.console (module), 194
robot.output.console.dotted (module), 195
robot.output.console.highlighting (module), 196
robot.output.console.quiet (module), 197
robot.output.console.verbose (module), 197
robot.output.debugfile (module), 198
robot.output.filelogger (module), 198
robot.output.librarylogger (module), 198
robot.output.listenerarguments (module), 199
robot.output.listenermethods (module), 199
robot.output.listener (module), 200
robot.output.logger (module), 200
robot.output.loggerhelper (module), 202
robot.output.output (module), 203
robot.output.pyloggingconf (module), 203
robot.output.stdoutlogsplitter (module), 205
robot.output.xmllogger (module), 205
robot.parsing (module), 206
robot.parsing.lexer (module), 207
robot.parsing.lexer.context (module), 207
robot.parsing.lexer.settings (module), 209
robot.parsing.lexer.tokens (module), 211
robot.parsing.model (module), 214
robot.parsing.model.blocks (module), 214
robot.parsing.model.statements (module), 217
robot.parsing.model.visitor (module), 237
robot.parsing.suitestructure (module), 237
robot.pythonpathsetter (module), 341
robot.rebot (module), 341
robot.reporting (module), 238
robot.reporting.expandkeywordmatcher (module), 238
robot.reporting.jsbuildingcontext (module), 238
robot.reporting.jsexecutionresult (module), 239
robot.reporting.jsmodelbuilders (module), 239
robot.reporting.logreportwriters (module), 240
robot.reporting.outputwriter (module), 240
robot.reporting.resultwriter (module), 242
robot.reporting.stringcache (module), 242
robot.reporting.xunitwriter (module), 243
robot.result (module), 245
robot.result.configurer (module), 246
robot.result.executionerrors (module), 247
robot.result.executionresult (module), 247
robot.result.flattenkeywordmatcher (module), 250
robot.result.keywordremover (module), 250
robot.result.merger (module), 256
robot.result.messagefilter (module), 257
robot.result.model (module), 258
robot.result.resultbuilder (module), 265
robot.result.suiteteardownfailed (module), 267
robot.result.visitor (module), 269
robot.result.xmlelementhandlers (module), 271
robot.run (module), 343
robot.running (module), 274
robot.running.arguments (module), 276
robot.running.arguments.argumentconverter (module), 276
robot.running.arguments.argumentmapper (module), 276
robot.running.arguments.argumentparser (module), 276
robot.running.arguments.argumentresolver (module), 277
robot.running.arguments.argumentresolver (module), 277
robot.running.arguments.argumentvalidator (module), 278
robot.running.arguments.embedded (module), 278
robot.running.arguments.typeconverters (module), 278
robot.running.arguments.typevalidator (module), 284
robot.running.builder (module), 284
robot.running.builder.builders (module), 284
robot.running.builder.parsers (module), 285
robot.running.builder.testsettings (module), 286
robot.running.builder.transformers (module), 286
robot.running.context (module), 289
robot.running.dynamicmethods (module), 290
robot.running.handler (module), 290
robot.running.handlerstore (module), 291
robot.running.importer (module), 291
robot.running.librarykeywordrunner (module), 291
robot.running.libraryscopes (module), 292
robot.running.model (module), 293
robot.running.namespace (module), 301
robot.running.outputcapture (module), 302
robot.running.randomizer (module), 302
robot.running.runkwregister (module), 303
robot.running.runner (module), 303
robot.running.signalhandler (module), 304
robot.running.status (module), 304
robot.running.statuserporter (module), 305
robot.running.steprunner (module), 305
robot.running.testlibraries (module), 306
robot.running.timeouts (module), 288
robot.running.timeouts.posix (module), 289
robot.running.timeouts.windows (module), 289
robot.running.userkeyword (module), 306
robot.running.userkeywordrunner (module), 307
robot.testdoc (module), 345
robot.tidy (module), 346
robot.tidypkg (module), 308
robot.tidypkg.transformers (module), 308
robot.utils (module), 310
robot.utils.application (module), 310
robot.utils.argumenterpaser (module), 311
robot.utils.asserts (module), 312
robot.utils.charset (module), 314
robot.utils.compat (module), 314
robot.utils.compress (module), 314
robot.utils.connectioncache (module), 314
robot.utils.dotdict (module), 315
robot.utils.encoding (module), 316
robot.utils.encodingsniffer (module), 317
robot.utils.error (module), 317
robot.utils.escaping (module), 317
robot.utils.etreewrapper (module), 317
robot.utils.filereader (module), 318
robot.utils.frange (module), 318
robot.utils.htmlformatters (module), 318
robot.utils.importer (module), 320
robot.utils.markuputils (module), 320
robot.utils.markupwriters (module), 321
robot.utils.match (module), 321
robot.utils.misc (module), 322
robot.utils.normalizing (module), 322
robot.utils.platform (module), 323
robot.utils.recommendations (module), 323
robot.utils.retester (module), 324
robot.utils.robotenv (module), 325
robot.utils.robotinspect (module), 325
robot.utils.robito (module), 325
robot.utils.robotpath (module), 325
robot.utils.robuttome (module), 326
robot.utils.robotypes (module), 327
robot.utils.robotypes2 (module), 327
robot.utils.setter (module), 328
robot.utils.sortable (module), 328
robot.utils.text (module), 328
robot.utils.unic (module), 328
robot.variables (module), 329
robot.variables.assigner (module), 329
robot.variables.evaluation (module), 329
robot.variables.filesetter (module), 330
robot.variables.finders (module), 330
robot.variables.isvar (module), 331
robot.variables.notfound (module), 331
robot.variables.replacer (module), 332
robot.variables.scopes (module), 332
robot.variables.search (module), 333
robot.variables.store (module), 334
robot.variables.tablesetter (module), 334
robot.variables.variables (module), 335
robot.version (module), 348
robot_handler_enabled() (in module robot.output.pyloggingconfig), 203

ROBOT_LIBRARY_SCOPE (robot.libraries.BuiltIn.BuiltIn attribute), 27

ROBOT_LIBRARY_SCOPE (robot.libraries.Collections.Collections attribute), 49

ROBOT_LIBRARY_SCOPE (robot.libraries.OperatingSystem.OperatingSystem attribute), 62


ROBOT_LIBRARY_SCOPE (robot.libraries.Remote.Remote attribute), 76

ROBOT_LIBRARY_SCOPE (robot.libraries.Reserved.Reserved attribute), 78

ROBOT_LIBRARY_SCOPE (robot.libraries.Screenshot.Screenshot attribute), 79

ROBOT_LIBRARY_SCOPE (robot.libraries.String.String attribute), 80

ROBOT_LIBRARY_SCOPE (robot.libraries.Telnet.Telnet attribute), 88

ROBOT_LIBRARY_SCOPE (robot.libraries.XML.XML attribute), 99

ROBOT_LIBRARY_VERSION (robot.libraries.BuiltIn.BuiltIn attribute), 27

ROBOT_LIBRARY_VERSION (robot.libraries.Collections.Collections attribute), 49

ROBOT_LIBRARY_VERSION (robot.libraries.OperatingSystem.OperatingSystem attribute), 62
attribute), 62
ROBOT_LIBRARY_VERSION
ROBOT_LIBRARY_VERSION
(robot.libraries.Screenshot.Screenshot attribute), 79
ROBOT_LIBRARY_VERSION
(robot.libraries.String.String attribute), 80
ROBOT_LIBRARY_VERSION
(robot.libraries.Telnet.Telnet attribute), 88
ROBOT_LIBRARY_VERSION
(robot.libraries.XML.XML attribute), 99
ROBOT_SUPPRESS_NAME
(robot.libraries.Telnet.NoMatchError attribute), 94
RobotDataStorage (class in robot.run), 324
RobotError, 335
RobotFramework (class in robot.run), 343
RobotHandler (class in robot.output.pyloggingconf), 203
RobotHandler (class in robot.result.xmlelementhandlers), 271
RobotModelWriter (class in robot.reporting.logreportwriters), 240
RobotNotRunningError, 47
RobotParser (class in robot.running.builder.parsers), 285
RobotSettings (class in robot.conf.settings), 18
RootHandler (class in robot.result.xmlelementhandlers), 271
roundup() (in module robot.misc), 322
rowconfigure() (robot.libraries.dialogs_py.InputDialog method), 125
rowconfigure() (robot.libraries.dialogs_py.MessageDialog and_return_rc() method), 112
rowconfigure() (robot.libraries.dialogs_py.MultilineSelectionDialog method), 151
rowconfigure() (robot.libraries.dialogs_py.PassFailDialog cli() (in module robot), 10
method), 164
rowconfigure() (robot.libraries.dialogs_py.SelectionDialog attribute), 18
run_keyword() (robot.libraries.BuiltIn.BuiltIn method), 36
run_keyword() (robot.libraries.Remote.Remote method), 77
run_keyword() (robot.libraries.XmlRpcRemoteClient method), 77
run_keyword() (robot.libraries.Reserved.Reserved method), 78
run_keyword_and_continue_on_failure() (robot.libraries.BuiltIn.BuiltIn method), 36
run() (in module robot.run), 344
run() (robot.running.librarykeywordrunner.EmbeddedArgumentsRunner method), 292
run() (robot.running.librarykeywordrunner.LibraryKeywordRunner method), 291
run() (robot.running.librarykeywordrunner.RunKeywordRunner method), 292
run() (robot.running.model.ForLoop method), 295
run() (robot.running.model.Keyword method), 293
run() (robot.running.model.TestSuite method), 298
run() (robot.running.steprunner.ForInEnumerateRunner method), 306
run() (robot.running.steprunner.ForInRangeRunner method), 306
run() (robot.running.steprunner.ForInRangeRunner method), 306
run() (robot.running.steprunner.InsembleArgumentsRunner method), 306
run() (robot.running.steprunner.InvalidForRunner method), 306
run() (robot.running.steprunner.UserErrorHandler method), 307
run() (robot.running.steprunner.UserKeywordRunner method), 307
run() (robot.running.timeouts.KeywordTimeout method), 289
run() (robot.running.timeouts.TimeoutException method), 289
run() (robot.running.usererrororhandler.UserErrorHandler method), 306
run() (robot.running.userkeywordrunner.EmbeddedArgumentsRunner method), 307
run() (robot.running.timeouts.TimeoutException method), 289
run_and_return_rc_and_output() (robot.libraries.OperatingSystem.OperatingSystem method), 62
run_cli() (in module robot.run), 343
run_empty_suite (robot.conf.settings.RobotSettings attribute), 18
RulerFormatter (class in robot.utils.htmlformatters), 318
run() (in module robot.run), 9
418 Index
selection_get() (robot.libraries.dialogs_py.InputDialog method), 125
selection_get() (robot.libraries.dialogs_py.MessageDialog method), 112
selection_get() (robot.libraries.dialogs_py.MultipleSelectionDialog method), 151
selection_get() (robot.libraries.dialogs_py.PassFailDialog method), 164
selection_get() (robot.libraries.dialogs_py.SelectionDialog method), 138
selection_handle() (robot.libraries.dialogs_py.InputDialog method), 126
selection_handle() (robot.libraries.dialogs_py.MessageDialog method), 112
selection_handle() (robot.libraries.dialogs_py.MultipleSelectionDialog method), 152
selection_handle() (robot.libraries.dialogs_py.PassFailDialog method), 165
selection_handle() (robot.libraries.dialogs_py.SelectionDialog method), 139
selection_handle() (robot.libraries.dialogs_py.SelectionDialog method), 132
send() (robot.libraries.dialogs_py.InputDialog method), 126
send() (robot.libraries.dialogs_py.MessageDialog method), 113
send() (robot.libraries.dialogs_py.MultipleSelectionDialog method), 152
send() (robot.libraries.dialogs_py.PassFailDialog method), 165
send() (robot.libraries.dialogs_py.SelectionDialog method), 139
send_content() (robot.libraries.Remote.TimeoutHTTPTransport method), 78
send_content() (robot.libraries.Remote.TimeoutHTTPSTransport method), 77
send_host() (robot.libraries.Remote.TimeoutHTTPSTransport method), 78
send_host() (robot.libraries.Remote.TimeoutHTTPTransport method), 77
send_request() (robot.libraries.Remote.TimeoutHTTPSTransport method), 78
send_request() (robot.libraries.Remote.TimeoutHTTPTransport method), 77
send_user_agent() (robot.libraries.Remote.TimeoutHTTPSTransport method), 78
send_user_agent() (robot.libraries.Remote.TimeoutHTTPTransport method), 77
SeparatorNormalizer (class in robot.parsing.lexer.tokens), 214
SeparatorNormalizer (class in robot.parsing.lexer.tokens), 212
set() (robot.variables.filesetter.VariableFileSetter method), 330
set() (robot.variables.tablesetter.VariableTableSetter method), 334
set_criticality() (robot.result.model.TestSuite method), 263
set() (robot.result.keywordremover.RemovalMessage method), 256
set() (robot.result.keywordremover.RemovalMessage method), 256
set_default_log_level() (robot.libraries.Telnet.TelnetConnection method), 90
set_earlier_failures() (robot.errors.ContinueForLoop method), 339
set_earlier_failures() (robot.errors.ContinueForLoop method), 339
set_earlier_failures() (robot.errors.ContinueForLoop method), 339
seq2str() (in module robot.utils.misc), 322
seq2str2() (in module robot.utils.misc), 322
set() (robot.variables.filesetter.VariableFileSetter method), 330
set() (robot.variables.tablesetter.VariableTableSetter method), 334
set_criticality() (robot.result.model.TestSuite method), 263
set_debuglevel() (robot.libraries.Telnet.TelnetConnection method), 94
set_default_log_level() (robot.libraries.Telnet.TelnetConnection method), 90
SelectionDialog (class in robot.libraries.dialogs_py), 139
SelectionDialog (class in robot.libraries.dialogs_py), 132
set() (robot.libraries.dialogs_py.InputDialog method), 126
SEPARATOR (robot.parsing.lexer.tokens.EOS attribute), 214
SEPARATOR (robot.parsing.lexer.tokens.Token attribute), 212
set_earlier_failures()
(robot.errors.ExecutionPassed method), 338
set_earlier_failures()
(robot.errors.ExitForLoop method), 339
set_earlier_failures()
(robot.errors.PassExecution method), 339
set_earlier_failures()
(robot.errors.ReturnFromKeyword method), 340
set_element_attribute()
(robot.libraries.XML.XML method), 103
set_element_tag()
(robot.libraries.XML.XML method), 103
set_element_text()
(robot.libraries.XML.XML method), 103
set_encoding()
(robot.libraries.Telnet.TelnetConnection method), 90
set_env_var()
(in module robot.utils.robotenv), 325
set_environment_variable()
(robot.libraries.OperatingSystem.OperatingSystem method), 67
set_error()
(robot.parsing.lexer.tokens.EOS method), 214
set_error()
(robot.parsing.lexer.tokens.Token method), 213
set_execution_mode()
(robot.result.executionresult.CombinedResult method), 249
set_execution_mode()
(robot.result.executionresult.Result method), 248
set_from_file()
(robot.variables.scopes.GlobalVariables method), 333
set_from_file()
(robot.variables.scopes.VariableScopes method), 332
set_from_file()
(robot.variables.variables.Variables method), 335
set_from_variable_table()
(robot.variables.scopes.GlobalVariables method), 333
set_from_variable_table()
(robot.variables.scopes.VariableScopes method), 332
set_from_variable_table()
(robot.variables.variables.Variables method), 335
set_global()
(robot.variables.scopes.SetVariables method), 333
set_global_variable()
(robot.libraries.BuiltIn.BuiltIn method), 39
set_element_attribute()
(robot.libraries.XML.XML method), 103
set_element_tag()
(robot.libraries.XML.XML method), 102
set_element_text()
(robot.libraries.XML.XML method), 102
set_element_attribute()
(robot.libraries.XML.XML method), 103
set_elements_tag()
(robot.libraries.XML.XML method), 102
set_elements_text()
(robot.libraries.XML.XML method), 103
set_encoding()
(robot.libraries.Telnet.TelnetConnection method), 90
set_env_var()
(in module robot.utils.robotenv), 325
set_environment_variable()
(robot.libraries.OperatingSystem.OperatingSystem method), 67
set_error()
(robot.parsing.lexer.tokens.EOS method), 214
set_error()
(robot.parsing.lexer.tokens.Token method), 213
set_execution_mode()
(robot.result.executionresult.CombinedResult method), 249
set_execution_mode()
(robot.result.executionresult.Result method), 248
set_from_file()
(robot.variables.scopes.GlobalVariables method), 333
set_from_file()
(robot.variables.scopes.VariableScopes method), 332
set_from_file()
(robot.variables.variables.Variables method), 335
set_from_variable_table()
(robot.variables.scopes.GlobalVariables method), 333
set_from_variable_table()
(robot.variables.scopes.VariableScopes method), 332
set_from_variable_table()
(robot.variables.variables.Variables method), 335
set_global()
(robot.variables.scopes.SetVariables method), 333
set_global_variable()
(robot.libraries.BuiltIn.BuiltIn method), 39
set_element_attribute()
(robot.libraries.XML.XML method), 103
set_element_tag()
(robot.libraries.XML.XML method), 102
set_element_text()
(robot.libraries.XML.XML method), 102
set_element_attribute()
(robot.libraries.XML.XML method), 103
set_elements_tag()
(robot.libraries.XML.XML method), 102
set_elements_text()
(robot.libraries.XML.XML method), 103
set_encoding()
(robot.libraries.Telnet.TelnetConnection method), 90
set_env_var()
(in module robot.utils.robotenv), 325
set_environment_variable()
(robot.libraries.OperatingSystem.OperatingSystem method), 67
set_error()
(robot.parsing.lexer.tokens.EOS method), 214
set_error()
(robot.parsing.lexer.tokens.Token method), 213
set_execution_mode()
(robot.result.executionresult.CombinedResult method), 249
set_execution_mode()
(robot.result.executionresult.Result method), 248
set_from_file()
(robot.variables.scopes.GlobalVariables method), 333
set_from_file()
(robot.variables.scopes.VariableScopes method), 332
set_from_file()
(robot.variables.variables.Variables method), 335
set_from_variable_table()
(robot.variables.scopes.GlobalVariables method), 333
set_from_variable_table()
(robot.variables.scopes.VariableScopes method), 332
set_from_variable_table()
(robot.variables.variables.Variables method), 335
set_global()
(robot.variables.scopes.SetVariables method), 333
set_global_variable()
(robot.libraries.BuiltIn.BuiltIn method), 39
set_if_removed()
(robot.result.keywordremover.RemovalMessage method), 256
set_keyword()
(robot.variables.scopes.SetVariables method), 333
set_keyword()
(robot.variables.scopes.VariableScopes method), 332
set_keyword_timeout()
(robot.running.timouts.TestTimeout method), 288
set_level()
(in module robot.output.pyloggingconf), 203
set_level()
(robot.output.filelogger.FileLogger method), 198
set_level()
(robot.output.logger.Logger method), 201
set_level()
(robot.output.loggerhelper.AbstractLogger method), 202
set_level()
(robot.output.loggerhelper.IsLogged method), 203
set_level()
(robot.output.output.Output method), 203
set_level()
(robot.output.outputwriter.OutputWriter method), 241
set_log_level()
(robot.libraries.BuiltIn.BuiltIn method), 39
set_log_level()
(robot.output.filelogger.FileLogger method), 198
set_log_level()
(robot.output.logger.Logger method), 201
set_log_level()
(robot.output.outputwriter.OutputWriter method), 241
set_log_level()
(robot.output.outputwriter.OutputWriter method), 241
set_modified_time()
(robot.libraries.OperatingSystem.OperatingSystem method), 69
set_name()
(robot.output.pyloggingconf.RobotHandler method), 205
set_newline() (robot.libraries.Telnet.TelnetConnection method), 89
set_option_negotiation_callback() (robot.libraries.Telnet.TelnetConnection method), 94
set_prompt() (robot.libraries.Telnet.TelnetConnection method), 90
set_screenshot_directory() (robot.libraries.Screenshot.Screenshot method), 79
set_search_order() (robot.running.namespace.Namespace method), 301
set_suite() (robot.variables.scopes.SetVariables method), 333
set_suite() (robot.variables.scopes.VariableScopes method), 332
set_suite_documentation() (robot.libraries.Builtin.Builtin method), 40
set_suite_metadata() (robot.libraries.Builtin.Builtin method), 40
set_suite_variable() (robot.libraries.Builtin.Builtin method), 42
set_tags() (robot.libraries.Builtin.Builtin method), 41
set_tags() (robot.model.testsuite.TestSuite method), 189
set_tags() (robot.result.model.TestSuite method), 265
set_tags() (robot.running.model.TestSuite method), 299
set_task_variable() (robot.libraries.Builtin.Builtin method), 41
set_telnetlib_log_level() (robot.libraries.Telnet.TelnetConnection method), 90
set_test() (robot.variables.scopes.SetVariables method), 333
set_test() (robot.variables.scopes.VariableScopes method), 332
set_test_documentation() (robot.libraries.Builtin.Builtin method), 41
set_test_message() (robot.libraries.Builtin.Builtin method), 41
set_test_variable() (robot.libraries.Builtin.Builtin method), 42
set_timeout() (robot.libraries.Telnet.TelnetConnection method), 89
set_to_dictionary() (robot.libraries.Collections.Collections method), 54
set_variable() (robot.libraries.Builtin.Builtin method), 42
set_variable_if() (robot.libraries.Builtin.Builtin method), 42
SetConverter (class in robot.running.arguments.typeconverters), 283
setdefault() (robot.model.metadata.Metadata method), 179
setdefault() (robot.utils.dotdict.DotDict method), 316
setdefault() (robot.utils.normalizing.NormalizedDict method), 323
setdefault() (robot.variables.evaluation.EvaluationNamespace method), 330
setFormatter() (robot.output.pyloggingconf.RobotHandler method), 204
setLevel() (robot.output.pyloggingconf.RobotHandler method), 204
setter (class in robot.utils.setter), 328
SetterAwareType (class in robot.utils.setter), 328
SETTING_HEADER (robot.parsing.lexer.tokens.EOS attribute), 214
SETTING_HEADER (robot.parsing.lexer.tokens.Token attribute), 212
Settings (class in robot.parsing.lexer.settings), 209
settings_class (robot.parsing.lexer.context.FileContext attribute), 208
settings_class (robot.parsing.lexer.context.InitFileContext attribute), 209
settings_class (robot.parsing.lexer.context.KeywordContext attribute), 209
settings_class (robot.parsing.lexer.context.LexingContext attribute), 207
settings_class (robot.parsing.lexer.context.ResourceFileContext attribute), 209
settings_class (robot.parsing.lexer.context.TestCaseFileContext attribute), 208
SETTING_TOKENS (robot.parsing.lexer.tokens.EOS attribute), 214
SETTING_TOKENS (robot.parsing.lexer.tokens.Token attribute), 212
Settings (class in robot.parsing.lexer.settings), 209
settings_class (robot.parsing.lexer.context.FileContext attribute), 208
settings_class (robot.parsing.lexer.context.InitFileContext attribute), 209
settings_class (robot.parsing.lexer.context.KeywordContext attribute), 209
settings_class (robot.parsing.lexer.context.LexingContext attribute), 207
settings_class (robot.parsing.lexer.context.ResourceFileContext.attribute), 208
settings_class (robot.parsing.lexer.context.TestCaseContext.attribute), 209
settings_class (robot.parsing.lexer.context.TestCaseFileContext.attribute), 208
SettingsBuilder (class in robot.running.builder.transformers), 286
SettingSection (class in robot.parsing.model.blocks), 215
SettingSectionHeader (class in robot.parsing.model.statements), 220
Setup (class in robot.parsing.model.statements), 230
setup (robot.model.keyword.Keywords attribute), 176
SETUP (robot.parsing.lexer.tokens.EOS attribute), 214
SETUP (robot.parsing.lexer.tokens.Token attribute), 212
setup (robot.running.builder.testsettings.TestDefaults attribute), 286
setup (robot.running.builder.testsettings.TestSettings attribute), 286
setup_executed() (robot.running.status.SuiteStatus method), 304
setup_executed() (robot.running.status.TestStatus method), 305
setup_message (robot.running.status.ParentMessage attribute), 305
setup_message (robot.running.status.SuiteMessage attribute), 305
setup_message (robot.running.status.TestMessage attribute), 305
SETUP_TYPE (robot.model.keyword.Keyword attribute), 175
SETUP_TYPE (robot.result.model.Keyword attribute), 260
SETUP_TYPE (robot.running.model.ForLoop attribute), 294
SETUP_TYPE (robot.running.model.Keyword attribute), 293
setvar() (robot.libraries.dialogs_py.InputDialog method), 126
setvar() (robot.libraries.dialogs_py.MessageDialog method), 113
setvar() (robot.libraries.dialogs_py.MultipleSelectionDialog method), 152
setvar() (robot.libraries.dialogs_py.PassFailDialog method), 165
setvar() (robot.libraries.dialogs_py.SelectionDialog method), 139
SetVariables (class in robot.variables.scopes), 333
severe() (robot.utils.restreader.CaptureRobotData method), 324
shortdoc (robot.libdocpkg.model.KeywordDoc attribute), 23
shortdoc (robot.running.usererrorhandler.UserErrorHandler), 306
shortdoc (robot.running.userkeyword.EmbeddedArgumentsHandler), 307
shortdoc (robot.running.userkeyword.UserKeywordHandler), 306
shortdoc (robot.running.userkeyword.UserKeywordHandler), 307
should_be_byte_string() (robot.libraries.String.String method), 84
should_be_empty() (robot.libraries.BuiltIn.BuiltIn method), 42
should_be_equal() (robot.libraries.BuiltIn.BuiltIn method), 42
should_be_equal_as_integers() (robot.libraries.BuiltIn.BuiltIn method), 43
should_be_equal_as_numbers() (robot.libraries.BuiltIn.BuiltIn method), 43
should_be_equal_as_strings() (robot.libraries.BuiltIn.BuiltIn method), 43
should_be_lowercase() (robot.libraries.String.String method), 85
should_be_uppercase() (robot.libraries.String.String method), 85
should_contain() (robot.libraries.BuiltIn.BuiltIn method), 44
should_contain_x_times() (robot.libraries.BuiltIn.BuiltIn method), 44
should_end_with() (robot.libraries.BuiltIn.BuiltIn method), 44
should_exist() (robot.libraries.OperatingSystem.OperatingSystem method), 63
should_match() (robot.libraries.BuiltIn.BuiltIn method), 44
should_not_be_empty() (robot.libraries.BuiltIn.BuiltIn method), 44
Index 423
should_not_be_equal()
(robot.libraries.BuiltIn.BuiltIn method), 45

should_not_be_equal_as_integers()
(robot.libraries.BuiltIn.BuiltIn method), 45

should_not_be_equal_as_numbers()
(robot.libraries.BuiltIn.BuiltIn method), 45

should_not_be_equal_as_strings()
(robot.libraries.BuiltIn.BuiltIn method), 45

should_not_be_string()
(robot.libraries.String.String method), 84

should_not_be_true()
(robot.libraries.BuiltIn.BuiltIn method), 45

should_not_contain()
(robot.libraries.BuiltIn.BuiltIn method), 46

should_not_contain_any()
(robot.libraries.BuiltIn.BuiltIn method), 46

should_not_contain_match()
(robot.libraries.Collections.Collections method), 49

should_not_end_with()
(robot.libraries.BuiltIn.BuiltIn method), 46

should_not_exist()
(robot.libraries.OperatingSystem.OperatingSystem method), 63

should_not_match()
(robot.libraries.BuiltIn.BuiltIn method), 46

should_not_match_regexp()
(robot.libraries.BuiltIn.BuiltIn method), 46

should_not_start_with()
(robot.libraries.BuiltIn.BuiltIn method), 46

should_start_with()
(robot.libraries.BuiltIn.BuiltIn method), 46

should_write_content_after_name()
(robot.tidypkg.transformers.ColumnAligner method), 309

show()
(robot.libdocpkg.consoleviewer.ConsoleViewer method), 22

show()
(robot.libraries.dialogs_py.InputDialog method), 126

show()
(robot.libraries.dialogs_py.MessageDialog method), 113

show()
(robot.libraries.dialogs_py.MultipleSelectionDialog method), 152

show()
(robot.libraries.dialogs_py.PassFailDialog method), 165

show()
(robot.libraries.dialogs_py.SelectionDialog method), 139

single_request()
(robot.libraries.Remote.TimeoutHTTPSTransport method), 78

single_request()
(robot.libraries.Remote.TimeoutHTTPTransport method), 77

single_value
(robot.parsing.lexer.settings.InitFileSettings attribute), 210

single_value
(robot.parsing.lexer.settings.KeywordSettings attribute), 211

SingleTagPattern (class in robot.model.tags), 185

SingleValue (class in robot.parsing.model.statements), 218

size()
(robot.libraries.dialogs_py.InputDialog method), 126

size()
(robot.libraries.dialogs_py.MessageDialog method), 113

size()
(robot.libraries.dialogs_py.MultipleSelectionDialog method), 152

skip_teardown_on_exit
(robot.conf.settings.RobotSettings attribute), 18

slaves()
(robot.libraries.dialogs_py.InputDialog method), 126

slaves()
(robot.libraries.dialogs_py.MessageDialog method), 113

slaves()
(robot.libraries.dialogs_py.MultipleSelectionDialog method), 152

slaves()
(robot.libraries.dialogs_py.PassFailDialog method), 165

slaves()
(robot.libraries.dialogs_py.SelectionDialog method), 139
slaves() (robot.libraries.dialogs_py.SelectionDialog method), 139
sleep() (robot.libraries.BuiltIn.BuiltIn method), 46
sock_avail() (robot.libraries.Telnet.TelnetConnection method), 94
sort () (robot.model.itemlist.ItemList method), 175
sort () (robot.model.keyword.Keywords method), 177
sort () (robot.model.message.Messages method), 178
sort () (robot.model.testcase.TestCases method), 188
sort () (robot.model.testsuite.Testsuites method), 191
sort () (robot.running.model.Imports method), 301
sort_list() (robot.libraries.Collections.Collections method), 54
Sortable (class in robot.utils.sortable), 328
source (robot.model.keyword.Keyword attribute), 176
source (robot.model.testcase_TestCase attribute), 188
source (robot.model.testsuite.TestSuite attribute), 189
source (robot.result.executionresult.Result attribute), 247
source (robot.result.keyword.Keyword attribute), 261
source (robot.result.model.TestCase attribute), 262
source (robot.result.model.TestSuite attribute), 265
source (robot.running.model.Keyword attribute), 295
source (robot.running.model.Keyword attribute), 296
source (robot.running.model.TestSuite attribute), 300
source (robot.running.model.UserKeyword attribute), 300
spacecount () (robot.tidy.ArgumentValidator method), 347
SpecDocBuilder (class in robot.libdocpkg.specbuilder), 24
split_args_from_name_or_path () (in module robot.utils.text), 328
split_command_line () (robot.libraries.Process.Process method), 75
split_extension () (robot.libraries.OperatingSystem.OperatingSystem method), 68
split_from_equals () (in module robot.escaping), 317
split_log (robot.conf.settings.RebotSettings attribute), 19
split_log (robot.conf.settings.RobotSettings attribute), 19
split_path() (robot.libraries.OperatingSystem.OperatingSystem method), 68
split_string () (robot.libraries.String.String method), 83
split_string_from_right () (robot.libraries.String.String method), 83
split_string_to_characters () (robot.libraries.String.String method), 83
split_tags_from_doc () (in module robot.utils.text), 328
split_to_lines () (robot.libraries.String.String method), 81
SplitLogWriter (class in robot.reporting.jswriter), 240
start () (robot.result.xmlelementhandlers.ArgumentsHandler method), 274
start () (robot.result.xmlelementhandlers.ArgumentsHandler method), 274
start () (robot.result.xmlelementhandlers.AssignHandler method), 273
start () (robot.result.xmlelementhandlers.KeywordHandler method), 271
start () (robot.result.xmlelementhandlers.KeywordStatusHandler method), 272
start () (robot.result.xmlelementhandlers.MessageHandler method), 272
start () (robot.result.xmlelementhandlers.MetadataHandler method), 272
start () (robot.result.xmlelementhandlers.MetaDataItemHandler method), 273
start () (robot.result.xmlelementhandlers.RootHandler method), 271
start () (robot.result.xmlelementhandlers.TagHandler method), 271
start () (robot.result.xmlelementhandlers.TagHandler method), 271
start () (robot.result.xmlelementhandlers.TagHandler method), 271
start () (robot.result.xmlelementhandlers.TestStatusHandler method), 272
start () (robot.running.timeouts.KeywordTimeout method), 273
start_directory (robot.running.builder.builders.SuiteStructureParser method), 285
start_directory (robot.tidy.Tidy method), 347
start_keyword (robot.conf.gatherfailed.GatherFailedTests method), 17
start_keyword (robot.conf.gatherfailed.GatherFailedSuites method), 16
start_keyword (robot.model.configurer.SuiteConfigurer method), 172
start_keyword (robot.model.filter.EmptySuiteRemover method), 173
start_keyword (robot.model.filter.Filter method), 174
start_keyword (robot.model.modifier.ModelModifier method), 180
start_keyword (robot.model.statistics.StatisticsBuilder method), 182
start_keyword (robot.model.tagsetter.TagSetter method), 186
start_keyword (robot.model.totalstatistics.TotalStatisticsBuilder method), 192
start_keyword (robot.model.visitor.SuiteVisitor method), 194
start_keyword (robot.output.console.dotted.StatusReporter message), 195
start_keyword (robot.output.console.verbose.StatusReporter message), 197
start_keyword (robot.output.filelogger.FileLogger method), 198
start_keyword (robot.output.logger.Logger method), 201
start_keyword (robot.output.output.Output method), 203
start_keyword (robot.output.xmllogger.XmlLogger method), 205
start_keyword (robot.reporting.outputwriter.OutputWriter method), 206
start_keyword (robot.reporting.xunitwriter.XUnitFileWriter method), 207
start_keyword (robot.result.merger.Merger method), 207
start_keyword (robot.result.messagefilter.MessageFilter method), 209
start_keyword (robot.result.resultbuilder.RemoveKeywords method), 210
start_keyword (robot.result.suiteteardownfailed.SuiteTeardownFailureHandler method), 210
start_keyword (robot.variables.scopes.SetVariables method), 212
start_keyword (robot.running.runner.Runner method), 214
start_keyword (robot.variables.scopes.VariableScopes method), 215
start_directory start_message (robot.running.timeouts.TestTimeout method), 289
start_directory start_message (robot.utils.markupwriters.HtmlWriter method), 321
start_directory start_message (robot.utils.markupwriters.NullMarkupWriter method), 321
start_directory start_message (robot.utils.markupwriters.XmlWriter method), 321
start_directory start_message (robot.reporting.xunitwriter.XUnitFileWriter method), 237
start_keyword start_message (robot.running.timeouts.TestTimeout method), 203
start_keyword start_message (robot.reporting.outputwriter.OutputWriter method), 201
start_keyword start_message (robot.reporting.xunitwriter.XUnitFileWriter method), 198
start_keyword start_message (robot.result.configurer.SuiteConfigurer method), 197
start_keyword start_message (robot.result.keywordremover.AllKeywordsRemover method), 195
start_keyword start_message (robot.result.keywordremover.ByTagKeywordRemover method), 194
start_keyword start_message (robot.result.keywordremover.ByNameKeywordRemover method), 192
start_keyword start_message (robot.result.keywordremover.OneByOneKeywordRemover method), 186
start_keyword start_message (robot.result.keywordremover.WaitUntilKeywordSucceedsRemover method), 182
start_keyword start_message (robot.result.keywordremover.PassedKeywordRemover method), 17
start_keyword start_message (robot.result.keywordremover.FailedKeywordRemover method), 17
start_keyword start_message (robot.result.keywordremover.UnexpectedKeywordRemover method), 17
start_keyword start_message (robot.result.keywordremover.WaitUntilKeywordSucceedsRemover method), 17
start_keyword start_message (robot.result.keywordremover.PassedKeywordRemover method), 17
start_keyword start_message (robot.result.keywordremover.FailedKeywordRemover method), 17
start_keyword start_message (robot.result.keywordremover.UnexpectedKeywordRemover method), 17
start_keyword start_message (robot.result.keywordremover.AllKeywordsRemover method), 17
start_keyword start_message (robot.variables.scopes.SetVariables method), 17
start_keyword start_message (robot.running.runner.Runner method), 17
start_keyword start_message (robot.variables.scopes.VariableScopes method), 17
start_keyword start_message (robot.running.runner.Runner method), 17
start_keyword start_message (robot.variables.scopes.SetVariables method), 17


start_message() (robot.model.modifier.ModelModifier method), 180
start_message() (robot.model.statistics.StatisticsBuilder method), 182
start_message() (robot.model.tagsetter.TagSetter method), 186
start_message() (robot.model.totalstatistics.TotalStatisticsBuilder method), 192
start_message() (robot.model.visitor.SuiteVisitor method), 194
start_message() (robot.output.console.dotted.StatusReporter method), 195
start_message() (robot.output.xmllogger.XmlLogger method), 206
start_message() (robot.result.merger.Merger method), 206
start_message() (robot.result.visitor.ResultVisitor method), 207
start_message() (robot.result.keywordremover.AllKeywordsRemover method), 208
start_message() (robot.result.keywordremover.ByNameKeywordRemover method), 209
start_message() (robot.result.keywordremover.ByTagKeywordRemover method), 210
start_message() (robot.result.keywordremover.PassedKeywordRemover method), 211
start_message() (robot.result.keywordremover.WaitUntilKeywordSucceedsRemover method), 212
start_message() (robot.result.keywordremover.ForLoopItemsRemover method), 213
start_message() (robot.result.keywordremover.AllKeywordsRemover method), 214
start_message() (robot.result.visitor.ResultVisitor method), 215
start_message() (robot.conf.gatherfailed.GatherFailedTests method), 216
start_message() (robot.conf.gatherfailed.GatherFailedSuites method), 217
start_message() (robot.model.modifier.ModelModifier method), 218
start_message() (robot.model.statistics.StatisticsBuilder method), 219
start_message() (robot.model.suitestatistics.SuiteStatisticsBuilder method), 220
start_message() (robot.model.modifier.ModelModifier method), 221
start_message() (robot.model.tagsetter.TagSetter method), 222
start_message() (robot.model.visitor.SuiteVisitor method), 223
start_message() (robot.running.randomizer.Randomizer method), 224
start_message() (robot.running.runner.Runner method), 225
start_result() (robot.output.xmllogger.XmlLogger method), 206
start_suite() (robot.output.console.verboselogging.Verboselogging method), 195
start_suite() (robot.output.console.verboselogging.VerboseloggingAntivirus method), 302
start_suite() (robot.output.console.verboselogging.VerboseloggingCore method), 302

start_suite() (robot.output.filelogger.FileLogger method), 197
start_suite() (robot.output.filelogger.FileLoggerAntivirus method), 303

start_suite() (robot.output.logger.Logger method), 201
start_suite() (robot.output.logger.LoggerAntivirus method), 303

start_suite() (robot.output.output.Output method), 203
start_suite() (robot.output.output.OutputAntivirus method), 305

start_suite() (robot.output.xmllogger.XmlLogger method), 205
start_suite() (robot.output.xmllogger.XmlLoggerAntivirus method), 305

start_suite() (robot.reporting.outputwriter.OutputWriter method), 241
start_suite() (robot.reporting.outputwriter.OutputWriterAntivirus method), 241

start_suite() (robot.reporting.xunitwriter.XUnitFileWriter method), 243
start_suite() (robot.reporting.xunitwriter.XUnitFileWriterAntivirus method), 244

start_suite() (robot.result.visitor.ResultVisitor method), 269
start_suite() (robot.result.visitor.ResultVisitorAntivirus method), 269

start_suite() (robot.running.runner.Runner method), 303
start_suite() (robot.running.runner.RunnerAntivirus method), 303

start_suite() (robot.variables.scopes.SetVariables method), 333
start_suite() (robot.variables.scopes.SetVariablesAntivirus method), 333
SuiteStatusHandler (class robot.result.xmlelementhandlers), 272
SuiteStructure (class robot.parsing.suitestructure), 237
SuiteStructureBuilder (class robot.parsing.suitestructure), 237
SuiteStructureParser (class robot.running.builder.builders), 284
SuiteStructureVisitor (class robot.parsing.suitestructure), 237
SuiteTeardown (class robot.parsing.model.statements), 226
SuiteTeardownFailed (class robot.result.suite teardownfailed), 268
SuiteTeardownFailureHandler (class robot.result.suite teardownfailed), 267
SuiteVisitor (class in robot.model.visitor), 193
SuiteWriter (class in robot.reporting.jswriter), 240
supports_kwargs (robot.result.xmlelementhandlers.ArgumentsHandler at-
robot.running.dynammethods.RunKeyword attribute), 290
switch() (robot.utils.connectioncache.ConnectionCache tag method), 315
switch_connection() (robot.libraries.Telnet.Telnet method), 89
system_decode() (in module robot.utils.encoding), 316
system_encode() (in module robot.utils.encoding), 316

T
TableFormatter (class in robot.utils.htmlformatters), 319
tag (robot.result.xmlelementhandlers.ArgumentHandler attribute), 274
tag (robot.result.xmlelementhandlers.ArgumentsHandler attribute), 274
tag (robot.result.xmlelementhandlers.AssignHandler attribute), 273
tag (robot.result.xmlelementhandlers.AssignVarHandler attribute), 273
tag (robot.result.xmlelementhandlers.DocHandler attribute), 272
tag (robot.result.xmlelementhandlers.ErrorsHandler attribute), 274
tag (robot.result.xmlelementhandlers.KeywordHandler attribute), 271
tag (robot.result.xmlelementhandlers.KeywordStatusHandler attribute), 272
tag (robot.result.xmlelementhandlers.MessageHandler attribute), 272
tag (robot.result.xmlelementhandlers.MetadataHandler attribute), 272
in tag (robot.result.xmlelementhandlers.MetadataItemHandler attribute), 273
in tag (robot.result.xmlelementhandlers.RobotHandler attribute), 271
in tag (robot.result.xmlelementhandlers.RootSuiteHandler attribute), 271
in tag (robot.result.xmlelementhandlers.StatisticsHandler attribute), 274
in tag (robot.result.xmlelementhandlers.SuiteHandler attribute), 271
in tag (robot.result.xmlelementhandlers.SuiteStatusHandler attribute), 272
in tag (robot.result.xmlelementhandlers.TagHandler attribute), 273
in tag (robot.result.xmlelementhandlers.TagsHandler attribute), 273
tag (robot.result.xmlelementhandlers.TestCaseHandler attribute), 271
tag (robot.result.xmlelementhandlers.TestStatusHandler attribute), 272
(tag (robot.result.xmlelementhandlers.TimeoutHandler attribute), 273
tag_is_critical() (robot.model.criticality.Criticality method), 172
tag_is_non_critical() (robot.model.criticality.Criticality method), 172
TagHandler (class in robot.result.xmlelementhandlers), 273
TagPattern() (in module robot.model.tags), 185
TagPatterns (class in robot.model.tags), 185
Tags (class in robot.model.tags), 185
Tags (class in robot.parsing.model.statements), 231
tags (robot.model.keyword.Keyword attribute), 176
tags (robot.model.statistics.Statistics attribute), 181
tags (robot.model.tagstatistics.TagStatistics attribute), 186
tags (robot.model.testcase.TestCase attribute), 187
TAGS (robot.parsing.lexer.tokens.EOS attribute), 214
TAGS (robot.parsing.lexer.tokens.Token attribute), 212
tags (robot.result.model.Keyword attribute), 261
tags (robot.result.model.TestCase attribute), 262
tags (robot.running.builder.testsettings.TestSettings attribute), 286
tags (robot.running.model.ForLoop attribute), 295
tags (robot.running.model.Keyword attribute), 294
tags (robot.running.model.TestCase attribute), 296
tagset (robot.running.model.UserKeyword attribute), 296
TagSetter (class in robot.model.tagsetter), 185
TagsHandler (class in robot.result.xmlelementhandlers), 273
TagStat (class in robot.model.stats), 183
TagStatDoc (class in robot.stats), 187
TagStatInfo (class in robot.model.tagstatistics), 187
TagStatistics (class in robot.model.tagstatistics), 186
TagStatisticsBuilder (class in robot.model.tagstatistics), 187
TagStatLink (class in robot.model.tagstatistics), 187
take_screenshot() (robot.libraries.Screenshot.Screenshot method), 79
take_screenshot_without_embedding() (robot.libraries.Screenshot.Screenshot method), 79
tasks (robot.parsing.model.blocks.TestCaseSection attribute), 215
Teardown (class in robot.parsing.model.statements), 230
teardown (robot.model.keyword.Keywords attribute), 177
TEARDOWN (robot.parsing.lexer.tokens.EOS attribute), 214
TEARDOWN (robot.parsing.lexer.tokens.Token attribute), 212
teardown_allowed (robot.running.status.Exit attribute), 304
teardown_allowed (robot.running.status_SUITEStatus attribute), 304
teardown_allowed (robot.running.status.TESTStatus attribute), 304
teardown_executed() (robot.running.status_SUITEStatus method), 304
teardown_executed() (robot.running.status_TESTStatus method), 305
teardown_message (robot.running.status.ParentMessage attribute), 305
teardown_message (robot.running.status_SUITEMessage attribute), 305
teardown_message (robot.running.status.TESTMessage attribute), 305
TEARDOWN_TYPE (robot.model.keyword.Keyword attribute), 175
TEARDOWN_TYPE (robot.result.model.Keyword attribute), 260
TEARDOWN_TYPE (robot.running.model.ForLoop attribute), 294
TEARDOWN_TYPE (robot.running.model.Keyword attribute), 293
Telnet (class in robot.libraries.Telnet), 85
TelnetConnection (class in robot.libraries.Telnet), 85
Template (class in robot.parsing.model.statements), 231
TEMPLATE (robot.parsing.lexer.tokens.EOS attribute), 214
TEMPLATE (robot.parsing.lexer.tokens.Token attribute), 212
template (robot.running.builder.testsettings.TestSettings attribute), 286
template (robot.running.model.TestCase attribute), 296
template_set (robot.parsing.lexer.context.KeywordContext attribute), 209
template_set (robot.parsing.lexer.context.TestCaseContext attribute), 209
template_set (robot.parsing.lexer.settings.TestCaseSettings attribute), 210
TemplateArguments (class in robot.parsing.model.statements), 234
TerminalEmulator (class in robot.libraries.Telnet), 94
terminate_all_processes() (robot.libraries.Process.Process method), 74
test() (robot.libraries.Screenshot.ScreenshotTaker method), 80
test_case_context() (robot.parsing.lexer.context.TestCaseFileContext method), 208
TEST_CASE_FILE_TYPE (robot.running.handlerstore.HandlerStore attribute), 291
test_case_file (robot.running.userkeyword.UserLibrary attribute), 307
test_case_section() (robot.parsing.lexer.context.FileContext method), 208
test_case_section() (robot.parsing.lexer.context.InitFileContext method), 209
test_case_section() (robot.parsing.lexer.context.ResourceFileContext method), 209
test_case_section() (robot.parsing.lexer.context.TestCaseFileContext method), 208
test_class (robot.model.testsuite.TestSuite attribute), 308
TestTimeout (class in robot.running.timeouts), 288
Tidy (class in robot.tidy), 346
tidy_cli() (in module robot.tidy), 347
TidyCommandLine (class in robot.tidy), 347
time_left() (robot.running.timeouts.KeywordTimeout
method), 289
time_left() (robot.running.timeouts.TestTimeout
method), 289
timed_out() (robot.running.timeouts.KeywordTimeout
method), 289
timed_out() (robot.running.timeouts.TestTimeout
method), 289
TimeDeltaConverter (class in robot.running.arguments.typeconverters),
281
Timeout (class in robot.parsing.model.statements), 232
Timeout (class in robot.running.timeouts.posix), 289
Timeout (class in robot.running.timeouts.windows),
289
timeout (robot.errors.ContinueForLoop attribute), 339
timeout (robot.errors.ExecutionFailed attribute), 337
timeout (robot.errors.ExecutionFailures attribute), 338
timeout (robot.errors.ExecutionPassed attribute), 338
timeout (robot.errors.ExecutionStatus attribute), 337
timeout (robot.errors.ExitForLoop attribute), 339
timeout (robot.errors.HandlerExecutionFailed
attribute), 337
timeout (robot.errors.PassExecution attribute), 339
timeout (robot.errors.ReturnFromKeyword attribute),
340
timeout (robot.errors.UserKeywordExecutionFailed
attribute), 338
timeout (robot.model.keyword.Keyword attribute), 176
timeout (robot.model.testcase.TestCase attribute), 187
TIMEOUT (robot.parsing.lexer.tokens.EOS attribute),
214
TIMEOUT (robot.parsing.lexer.tokens.Token attribute),
212
timeout (robot.result.model.Keyword attribute), 261
timeout (robot.result.model.TestCase attribute), 262
timeout (robot.running.builder.testsettings.TestDefaults
attribute), 286
timeout (robot.running.builder.testsettings.TestSettings
attribute), 286
timeout (robot.running.model.ForLoop attribute), 295
timeout (robot.running.model.Keyword attribute), 294
timeout (robot.running.model_TestCase attribute),
296
TimeoutError, 336
TimeoutHandler (class in robot.running.arguments.typehandlers), 273
TimeoutHTTPTransport (class in robot.libraries.Remote), 77
TimeoutHTTPTransport (class in robot.libraries.Remote), 77
timestamp (robot.model.message.Message attribute),
177
timestamp (robot.output.loggerhelper.Message attribute), 203
timestamp (robot.result.model.Message attribute), 259
timestamp() (robot.reporting.jsbuildingcontext.JsBuildingContext
method), 238
timestamp_to_secs() (in module robot.utils.robottime), 326
TimestampCache (class in robot.utils.robottime), 327
timestr_to_secs() (in module robot.utils.robottime), 326
title() (robot.libraries.dialogs_py.InputDialog
method), 126
title() (robot.libraries.dialogs_py.MessageDialog
method), 113
title() (robot.libraries.dialogs_py.MultilineSelectionDialog
method), 152
title() (robot.libraries.dialogs_py.PassFailDialog
method), 165
title() (robot.libraries.dialogs_py.SelectionDialog
method), 139
tk_bisque() (robot.libraries.dialogs_py.InputDialog
method), 126
tk_bisque() (robot.libraries.dialogs_py.MessageDialog
method), 113
tk_bisque() (robot.libraries.dialogs_py.MultilineSelectionDialog
method), 152
tk_bisque() (robot.libraries.dialogs_py.PassFailDialog
method), 165
tk_bisque() (robot.libraries.dialogs_py.SelectionDialog
method), 139
tk_focusFollowsMouse() (robot.libraries.dialogs_py.InputDialog
method), 126
tk_focusFollowsMouse() (robot.libraries.dialogs_py.MessageDialog
method), 113
tk_focusFollowsMouse() (robot.libraries.dialogs_py.MultilineSelectionDialog
method), 152
tk_focusFollowsMouse() (robot.libraries.dialogs_py.PassFailDialog
method), 165
tk_focusNext() (robot.libraries.dialogs_py.InputDialog
method), 126
tk_focusNext() (robot.libraries.dialogs_py.MessageDialog
method), 113
tk_focusNext() (robot.libraries.dialogs_py.MultilineSelectionDialog
method), 152
tk_focusNext() (robot.libraries.dialogs_py.PassFailDialog
method), 165
method), 165
tk_focusNext () (robot.libraries.dialogs_py.SelectionDialog method), 139
tk_focusPrev () (robot.libraries.dialogs_pyInputDialog total (robot.model.stats.CombinedTagStat attribute), 184
tk_focusPrev () (robot.libraries.dialogs_pyInputDialog total (robot.model.stats.CriticalTagStat attribute), 184
tk_focusPrev () (robot.libraries.dialogs_pyInputDialog total (robot.model.stats.TagStat attribute), 184
tk_focusPrev () (robot.libraries.dialogs_pyInputDialog total (robot.model.stats.TotalStat attribute), 183
TotalStat (class in robot.model.stats), 182
tk_focusPrev () (robot.libraries.dialogs_pyInputDialog TotalStatistics (class in robot.model.stats), 191
TotalStatisticsBuilder (class in robot.model.stats), 191
tk_menuBar () (robot.libraries.dialogs_pyInputDialog (robot.libraries.OperatingSystem.OperatingSystem method), 70
method), 113
tk_menuBar () (robot.libraries.dialogs_pyInputDialog (in module robot.api.logger) (module robot.output.logger), 198
method), 153
tk_menuBar () (robot.libraries.dialogs_pyInputDialog trace () (robot.output.filelogger.FileLogger method), 198
method), 166
tk_menuBar () (robot.libraries.dialogs_pyInputDialog trace () (robot.output.logger.Logger method), 201
method), 140
tk_menuBar () (robot.libraries.dialogs_pyInputDialog trace () (robot.output.loggerhelper.AbstractLogger
method), 202
tk_menuBar () (robot.libraries.dialogs_pyInputDialog (in module robot.output.Output method), 203
method), 140
tk_menuBar () (robot.libraries.dialogs_pyInputDialog traceback () (robot.utils.error.JavaErrorDetails
method), 113
attribute), 317
tk_menuBar () (robot.libraries.dialogs_pyInputDialog traceback () (robot.utils.error.PythonErrorDetails
method), 153
attribute), 317
tk_menuBar () (robot.libraries.dialogs_pyInputDialog (robot.libraries.OperatingSystem.OperatingSystem
method), 166
tk_menuBar () (robot.libraries.dialogs_pyInputDialog (class in robot.running.arguments.typeconverters),
method), 127
tk_menuBar () (robot.libraries.dialogs_pyInputDialog (class in robot.running.context.ExecutionContexts
method), 114
tk_menuBar () (robot.libraries.dialogs_pyInputDialog (class in robot.running.context.ExecutionContexts
method), 114
tk_strictMotif () (robot.libraries.dialogs_pyInputDialog (class in robot.running.context.ExecutionContexts
method), 114
tk_strictMotif () (robot.libraries.dialogs_pyInputDialog (class in robot.running.context.ExecutionContexts
method), 114
tk_strictMotif () (robot.libraries.dialogs_pyInputDialog (class in robot.running.context.ExecutionContexts
method), 114
tk_strictMotif () (robot.libraries.dialogs_pyInputDialog (class in robot.running.context.ExecutionContexts
method), 114
tk_strictMotif () (robot.libraries.dialogs_pyInputDialog (class in robot.running.context.ExecutionContexts
method), 114
tk_strictMotif () (robot.libraries.dialogs_pyInputDialog (class in robot.running.context.ExecutionContexts
method), 114
tk_strictMotif () (robot.libraries.dialogs_pyInputDialog (class in robot.running.context.ExecutionContexts
method), 114
tk_strictMotif () (robot.libraries.dialogs_pyInputDialog (class in robot.running.context.ExecutionContexts
method), 114
tk_strictMotif () (robot.libraries.dialogs_pyInputDialog (class in robot.running.context.ExecutionContexts
method), 114
Tkraise () (robot.libraries.dialogs_pyInputDialog TupleListDumper (class in robot.html.data.json_writer), 21
(method), 127
tkraise () (robot.libraries.dialogs_pyInputDialog type (robot.model.keyword.Keyword attribute), 176
tkraise () (robot.libraries.dialogs_pyInputDialog type (robot.model.stats.CombinedTagStat attribute), 184
method), 114
Tkraise () (robot.libraries.dialogs_pyInputDialog type (robot.model.stats.SuiteStat attribute), 183
tkraise () (robot.libraries.dialogs_pyInputDialog type (robot.model.stats.TagStat attribute), 183
tkraise () (robot.libraries.dialogs_pyInputDialog type (robot.model.stats.TotalStat attribute), 183
tkraise () (robot.libraries.dialogs_pyInputDialog type (robot.parsing.lexer.tokens.EOS attribute), 214
tkraise () (robot.libraries.dialogs_pyInputDialog type (robot.parsing.lexer.tokens.Token attribute), 212
method), 140
Token (class in robot.parsing.lexer.tokens), 211
type (robot.parsing.model.statements.Arguments

Index 435
type (robot.parsing.model.statements.Comment attribute), 232
type (robot.parsing.model.statements.CommentSectionHeader attribute), 255
type (robot.parsing.model.statements.DefaultTags attribute), 233
type (robot.parsing.model.statements.DocumentationOrMetadata attribute), 218
type (robot.parsing.model.statements.Documentation attribute), 218
type (robot.parsing.model.statements.EmptyLine attribute), 236
type (robot.parsing.model.statements.End attribute), 235
type (robot.parsing.model.statements.Error attribute), 236
type (robot.parsing.model.statements.Fixture attribute), 219
type (robot.parsing.model.statements.ForceTags attribute), 225
type (robot.parsing.model.statements.ForLoopHeader attribute), 234
type (robot.parsing.model.statements.KeywordCall attribute), 233
type (robot.parsing.model.statements.KeywordName attribute), 229
type (robot.parsing.model.statements.KeywordSectionHeader attribute), 221
type (robot.parsing.model.statements.LibrarySectionHeader attribute), 222
type (robot.parsing.model.statements.LibraryImport attribute), 222
type (robot.parsing.model.statements.LibrarySectionHeader attribute), 222
type (robot.parsing.model.statements.LibraryImport attribute), 222
type (robot.parsing.model.statements.MultiValue attribute), 219
type (robot.parsing.model.statements.Setting attribute), 224
type (robot.parsing.model.statements.Setting attribute), 224
type (robot.parsing.model.statements.Statement attribute), 225
type (robot.parsing.model.statements.Statement attribute), 225
type (robot.parsing.model.statements.Statement attribute), 225
type (robot.parsing.model.statements.Statement attribute), 225
type (robot.parsing.model.statements.Statement attribute), 225
type (robot.parsing.model.statements.Statement attribute), 225
type (robot.parsing.model.statements.Tags attribute), 226
type (robot.parsing.model.statements.Tags attribute), 226
type (robot.parsing.model.statements.Tags attribute), 226
type (robot.parsing.model.statements.Tags attribute), 226
type (robot.parsing.model.statements.Tags attribute), 230
type (robot.parsing.model.statements.Template attribute), 231
type (robot.parsing.model.statements.Teardown attribute), 230
type (robot.parsing.model.statements.Teardown attribute), 230
type (robot.parsing.model.statements.TestCaseName attribute), 241
type (robot.parsing.model.statements.TestCaseSectionHeader attribute), 221
type (robot.parsing.model.statements.TestSetup attribute), 227
type (robot.parsing.model.statements.TestTeardown attribute), 227
type (robot.parsing.model.statements.TestTemplate attribute), 228
type (robot.parsing.model.statements.Timeout attribute), 228
type (robot.parsing.model.statements.Typed Teardown attribute), 232
type (robot.parsing.model.statements.Variable attribute), 228
type (robot.parsing.model.statements.VariableSectionHeader attribute), 220
type (robot.parsing.model.statements.VariablesImport attribute), 223
type (robot.running.arguments.typeconverters.BooleanConverter attribute), 278
type (robot.running.arguments.typeconverters.ByteStringConverter attribute), 280
type (robot.running.arguments.typeconverters.BytesConverter attribute), 280
type (robot.running.arguments.typeconverters.DateTimeConverter attribute), 283
type (robot.running.arguments.typeconverters.DecimalConverter attribute), 279
type (robot.running.arguments.typeconverters.DictionaryConverter attribute), 283
type (robot.running.arguments.typeconverters.DecimalConverter attribute), 279
type (robot.running.arguments.typeconverters.DateTimeConverter attribute), 283
type (robot.running.arguments.typeconverters.DecimalConverter attribute), 279
type (robot.running.arguments.typeconverters.DateTimeConverter attribute), 283
type (robot.running.arguments.typeconverters.DecimalConverter attribute), 279
type (robot.running.arguments.typeconverters.SetConverter TypeValidator) (class in robot.running.arguments.typeconverters), 283

type (robot.running.arguments.typeconverters.TimeDeltaConverter attribute), 284

type (robot.running.arguments.typeconverters.TupleConverter attribute), 282

unbind() (robot.libraries.dialogs_py.InputDialog method), 127

unbind() (robot.libraries.dialogs_py.MessageDialog method), 114

unbind() (robot.libraries.dialogs_py.SelectionDialog method), 140

unbind_all() (robot.libraries.dialogs_py.InputDialog method), 127

unbind_all() (robot.libraries.dialogs_py.MessageDialog method), 114

unbind_all() (robot.libraries.dialogs_py.SelectionDialog method), 153

unbind_all() (robot.libraries.dialogs_py.PassFailDialog method), 166

unbind_all() (robot.libraries.dialogs_py.MultipleSelectionDialog method), 153

unbind_all() (robot.libraries.dialogs_py.P televised selection Dialog method), 166

unbind_all() (robot.libraries.dialogs_py.SelectionDialog method), 140

unbind_class() (robot.libraries.dialogs_py.InputDialog method), 127

unbind_class() (robot.libraries.dialogs_py.MessageDialog method), 114

unbind_class() (robot.libraries.dialogs_py.SelectionDialog method), 153

unbind_class() (robot.libraries.dialogs_py.PassFailDialog method), 166

unbind_class() (robot.libraries.dialogs_py.MultipleSelectionDialog method), 153

unbind_class() (robot.libraries.dialogs_py.P televised selection Dialog method), 166

unbind_class() (robot.libraries.dialogs_py.SelectionDialog method), 140

unescape() (robot.util.escaping. Unescaper method), 334

unescape_variable_syntax() (in module robot.variables.search), 334

Unescaper (class in robot.util.escaping), 317

unbind_all() (robot.util.escaping, Unescaper method), 334

unescape_variable_syntax() (in module robot.util.escaping, Unescaper method), 334

unbind_all() (robot.util.escaping, Unescaper method), 317

unbind_class() (robot.util.escaping, Unescaper method), 328

unregister_logger() (robot.output.logger.Logger method), 201

unregister_logger() (robot.output.logger.Logger method), 201

unregister() (robot.output.listeners.LibraryListeners method), 200

unregister() (robot.output.listeners.LibraryListeners method), 200

unregister_console_logger() (robot.output.listeners.LibraryListeners method), 200

unlink_converter() (robot.output.logger.Logger method), 201

unlink_converter() (robot.output.logger.Logger method), 201

unnest() (robot.util.escaping, Unescaper method), 105

unbind_all() (robot.output.logger.Logger method), 201

unbind_all() (robot.output.logger.Logger method), 201

unescape_variable_syntax() (in module robotutil.types2), 327

unbind() (robot.libraries.dialogs_py.InputDialog method), 127

unbind() (robot.libraries.dialogs_py.MessageDialog method), 114

unbind() (robot.libraries.dialogs_py.MultipleSelectionDialog method), 153

unbind() (robot.libraries.dialogs_py.PassFailDialog method), 166

unbind_all() (robot.libraries.dialogs_py.SelectionDialog method), 140

unbind_all() (robot.libraries.dialogs_py.SelectionDialog method), 140

unbind_class() (robot.libraries.dialogs_py.InputDialog method), 127

unbind_class() (robot.libraries.dialogs_py.MessageDialog method), 114

unbind_class() (robot.libraries.dialogs_py.SelectionDialog method), 153

unbind_class() (robot.libraries.dialogs_py.PassFailDialog method), 166

unbind_class() (robot.libraries.dialogs_py.MultipleSelectionDialog method), 153

unbind_class() (robot.libraries.dialogs_py.SelectivaDia
gen method), 166

unbind_class() (robot.libraries.dialogs_py.SelectionDialog method), 140

unescape() (robot.util.escaping, Unescaper method), 334

unescape_variable_syntax() (in module robot.util.escaping), 317

unregister_listener() (robot.output.listeners.LibraryListeners method), 200

unregister_logger() (robot.output.logger.Logger method), 201

unregister() (robot.output.listeners.LibraryListeners method), 200

unregister() (robot.output.listeners.LibraryListeners method), 200

unregister_console_logger() (robot.output.listeners.LibraryListeners method), 200

unlink_converter() (robot.output.logger.Logger method), 201

unlink_converter() (robot.output.logger.Logger method), 201

unescape() (robot.util.escaping, Unescaper method), 105

unbind_all() (robot.output.logger.Logger method), 201

unbind_all() (robot.output.logger.Logger method), 201

unescape_variable_syntax() (in module robotutil.types2), 327
Index
visit() (robot.tidypkg.transformers.NewLineNormalizer method), 308
visit() (robot.tidypkg.transformers.SeparatorNormalizer method), 309
visit_Arguments() (robot.running.builder.transformers.KeywordBuilder method), 288
visit_CommentSection() (robot.tidypkg.transformers.Cleaner method), 308
visit_CommentSection() (robot.tidypkg.transformers.NewLineNormalizer method), 308
visit_DefaultTags() (robot.running.builder.transformers.SettingsBuilder method), 286
visit_directory() (robot.parsing.suitestructure.SuiteStructureVisitor), 287
visit_directory() (robot.running.builder.builders.SuiteStructureParser), 285
visit_directory() (robot.running.builder.builders.SuiteStructureParser method), 174
visit_directory() (robot.tidy.Tidy method), 347
visit_Documentation() (robot.running.builder.transformers.KeywordBuilder method), 288
visit_Documentation() (robot.running.builder.transformers.TestCaseBuilder method), 287
visit_Error() (robot.running.builder.parsers.ErrorReporter method), 285
visit_errors() (robot.output.xmllogger.XmlLogger method), 206
visit_errors() (robot.running.builder.builders.SuiteStructureParser method), 285
visit_file() (robot.tidy.Tidy method), 347
visit_File() (robot.tidypkg.transformers.NewLineNormalizer method), 308
visit_ForceTags() (robot.running.builder.transformers.SettingsBuilder method), 286
visit_suite() (robot.result.keywordremover.ByNameKeywordRemover method), 251
visit_tag_statistics() (robot.reporting.xunitwriter.XUnitFileWriter method), 245
visit_suite() (robot.result.keywordremover.ByTagKeywordRemover method), 253
visit_suite() (robot.result.keywordremover.ForLoopItemsRemover method), 254
visit_suite() (robot.result.keywordremover.PassedKeywordRemover method), 252
visit_suite() (robot.result.keywordremover.WaitUntilKeywordSucceedsRemover method), 255
visit_suite() (robot.result.keywordremover.WarningAndErrorFinder method), 256
visit_suite() (robot.result.merger.Merger method), 257
visit_suite() (robot.result.messagefilter.MessageFilter method), 258
visit_suite() (robot.result.suiteteardownfailed.SuiteTeardownFailed method), 269
visit_suite() (robot.result.suiteteardownfailed.SuiteTeardownFailureHandler method), 268
visit_suite() (robot.result.visitor.ResultVisitor method), 270
visit_suite() (robot.running.randomizer.Randomizer method), 303
visit_suite() (robot.running.builder.transformers.SettingsBuilder method), 286
visit_suite() (robot.running.builder.transformers.TestCaseBuilder method), 287
visit_suite() (robot.running.builder.transformers.KeywordBuilder method), 288
visit_suite() (robot.running.builder.transformers.TestCaseBuilder method), 287
visit_suite() (robot.running.builder.transformers.ForLoopBuilder method), 288
visit_suite() (robot.running.builder.transformers.TestCaseBuilder method), 287
visit_suite() (robot.conf.gatherfailed.GatherFailedTests method), 16
visit_suite() (robot.conf.gatherfailed.GatherFailedSuites method), 17
visit_suite() (robot.output.xmllogger.XmlLogger method), 206
visit_suite() (robot.output.xmllogger.XmlLogger method), 206
visit_suite() (robot.output.xmllogger.XmlLogger method), 206
visit_suite() (robot.output.xmllogger.XmlLogger method), 206
visit_suite() (robot.running.builder.transformers.SettingsBuilder method), 286
visit_suite() (robot.running.builder.transformers.SettingsBuilder method), 286
visit_suite() (robot.running.builder.transformers.ForLoopBuilder method), 288
visit_suite() (robot.running.builder.transformers.TestCaseBuilder method), 287
visit_suite() (robot.output.console.dotted.StatusReporter method), 195
visit_suite() (robot.output.xmllogger.XmlLogger method), 206
visit_suite() (robot.reporting.outputwriter.OutputWriter method), 242
visit_suite() (robot.reporting.outputwriter.OutputWriter method), 242
visit_suite() (robot.reporting.xunitwriter.XUnitFileWriter method), 243
visit_suite() (robot.result.visitor.ResultVisitor method), 269
visit_suite() (robot.result.visitor.ResultVisitor method), 269
visit_suite() (robot.result.visitor.ResultVisitor method), 269
Index 445
wait_variable (robot.libraries.dialogs_py.MessageDialog method), 114
wait_variable (robot.libraries.dialogs_py.MultilineSelectionDialog method), 153
wait_variable (robot.libraries.dialogs_py.PassFailDialog method), 166
wait_variable (robot.libraries.dialogs_py.SelectionDialog method), 140
wait_visibility (robot.libraries.dialogs_py.InputDialog method), 127
wait_visibility (robot.libraries.dialogs_py.MessageDialog method), 114
wait_visibility (robot.libraries.dialogs_py.MultilineSelectionDialog method), 153
wait_visibility (robot.libraries.dialogs_py.PassFailDialog method), 166
wait_visibility (robot.libraries.dialogs_py.SelectionDialog method), 140
wait_window (robot.libraries.dialogs_py.InputDialog method), 127
wait_window (robot.libraries.dialogs_py.MessageDialog method), 114
wait_window (robot.libraries.dialogs_py.MultilineSelectionDialog method), 153
wait_window (robot.libraries.dialogs_py.PassFailDialog method), 166
wait_window (robot.libraries.dialogs_py.SelectionDialog method), 140
waiting_item_state (robot.variables.search.VariableSearcher method), 334
WaitUntilKeywordSucceedsRemover (class in robot.result.keywordremover), 254
waitvar (robot.libraries.dialogs_py.InputDialog method), 127
waitvar (robot.libraries.dialogs_py.MessageDialog method), 114
waitvar (robot.libraries.dialogs_py.MultilineSelectionDialog method), 153
waitvar (robot.libraries.dialogs_py.PassFailDialog method), 166
waitvar (robot.libraries.dialogs_py.SelectionDialog method), 140
warn (robot.output.output.Output method), 203
warn (robot.utilis.restreader.CaptureRobotData method), 324
warnAndErrorFinder (class in robot.result.keywordremover), 255
winfo_atom (robot.libraries.dialogs_py.InputDialog method), 127
winfo_atom (robot.libraries.dialogs_py.MessageDialog method), 114
winfo_atom (robot.libraries.dialogs_py.MultilineSelectionDialog method), 154
winfo_atom (robot.libraries.dialogs_py.PassFailDialog method), 167
winfo_atom (robot.libraries.dialogs_py.SelectionDialog method), 141
winfo_atom (robot.libraries.dialogs_py.InputDialog method), 127
winfo_atom (robot.libraries.dialogs_py.MessageDialog method), 114
winfo_atom (robot.libraries.dialogs_py.MultilineSelectionDialog method), 154
winfo_atom (robot.libraries.dialogs_py.PassFailDialog method), 167
winfo_atom (robot.libraries.dialogs_py.SelectionDialog method), 141
widths_for_line (robot.tidypkg.transformers.ColumnAligner method), 309
robot.output.logger.Logger method, 202
robot.output.librarylogger Logger method, 198
robot.output.filelogger.FileLogger method, 198
robot.output.loggerhelper(AbstractLogger method), 201
robot.output.librarylogger Logger method, 198
robot.output.filelogger.FileLogger method, 198
robot.output.loggerhelper(AbstractLogger method), 201
robot.output.librarylogger Logger method, 198
robot.output.filelogger.FileLogger method, 198
robot.output.loggerhelper(AbstractLogger method), 201
`winfo_ismapped()` (robot.libraries.dialogs_py.MultipleSelectionDialog method), 154
`winfo_ismapped()` (robot.libraries.dialogs_py.PassFailDialog method), 167
`winfo_ismapped()` (robot.libraries.dialogs_py.SelectionDialog method), 141
`winfo_manager()` (robot.libraries.dialogs_py.InputDialog method), 128
`winfo_manager()` (robot.libraries.dialogs_py.MessageDialog method), 115
`winfo_manager()` (robot.libraries.dialogs_py.MultipleSelectionDialog method), 154
`winfo_manager()` (robot.libraries.dialogs_py.PassFailDialog method), 167
`winfo_manager()` (robot.libraries.dialogs_py.SelectionDialog method), 141
`winfo_name()` (robot.libraries.dialogs_py.InputDialog method), 128
`winfo_name()` (robot.libraries.dialogs_py.MessageDialog method), 115
`winfo_name()` (robot.libraries.dialogs_py.MultipleSelectionDialog method), 154
`winfo_name()` (robot.libraries.dialogs_py.PassFailDialog method), 167
`winfo_name()` (robot.libraries.dialogs_py.SelectionDialog method), 141
`winfo_parent()` (robot.libraries.dialogs_py.InputDialog method), 128
`winfo_parent()` (robot.libraries.dialogs_py.MessageDialog method), 115
`winfo_parent()` (robot.libraries.dialogs_py.MultipleSelectionDialog method), 154
`winfo_parent()` (robot.libraries.dialogs_py.PassFailDialog method), 167
`winfo_parent()` (robot.libraries.dialogs_py.SelectionDialog method), 141
`winfo_pathname()` (robot.libraries.dialogs_py.InputDialog method), 128
`winfo_pathname()` (robot.libraries.dialogs_py.MessageDialog method), 115
`winfo_pathname()` (robot.libraries.dialogs_py.MultipleSelectionDialog method), 154
`winfo_pathname()` (robot.libraries.dialogs_py.PassFailDialog method), 167
`winfo_pathname()` (robot.libraries.dialogs_py.SelectionDialog method), 141
`winfo_pixels()` (robot.libraries.dialogs_py.InputDialog method), 128
`winfo_pixels()` (robot.libraries.dialogs_py.MessageDialog method), 115
`winfo_pixels()` (robot.libraries.dialogs_py.MultipleSelectionDialog method), 154
`winfo_pixels()` (robot.libraries.dialogs_py.PassFailDialog method), 169
`winfo_pixels()` (robot.libraries.dialogs_py.SelectionDialog method), 141
`winfo_pointerx()` (robot.libraries.dialogs_py.InputDialog method), 128
`winfo_pointerx()` (robot.libraries.dialogs_py.MessageDialog method), 115
`winfo_pointerx()` (robot.libraries.dialogs_py.MultipleSelectionDialog method), 154
`winfo_pointerx()` (robot.libraries.dialogs_py.PassFailDialog method), 167
`winfo_pointerx()` (robot.libraries.dialogs_py.SelectionDialog method), 141
`winfo_pointery()` (robot.libraries.dialogs_py.InputDialog method), 128
`winfo_pointery()` (robot.libraries.dialogs_py.MessageDialog method), 115
`winfo_pointery()` (robot.libraries.dialogs_py.MultipleSelectionDialog method), 154
`winfo_pointery()` (robot.libraries.dialogs_py.PassFailDialog method), 167
`winfo_pointery()` (robot.libraries.dialogs_py.SelectionDialog method), 141
`winfo_reqheight()` (robot.libraries.dialogs_py.InputDialog method), 129
`winfo_reqheight()` (robot.libraries.dialogs_py.MessageDialog method), 115
`winfo_reqheight()` (robot.libraries.dialogs_py.MultipleSelectionDialog method), 155
`winfo_reqheight()` (robot.libraries.dialogs_py.PassFailDialog method), 168
`winfo_reqheight()` (robot.libraries.dialogs_py.SelectionDialog method), 142
`winfo_reqwidth()` (robot.libraries.dialogs_py.InputDialog method), 132
`winfo_reqwidth()` (robot.libraries.dialogs_py.MessageDialog method), 115
`winfo_reqwidth()` (robot.libraries.dialogs_py.MultipleSelectionDialog method), 155
`winfo_reqwidth()` (robot.libraries.dialogs_py.PassFailDialog method), 168
`winfo_reqwidth()` (robot.libraries.dialogs_py.SelectionDialog method), 142
winfo_visualsavailable()  
  (robot.libraries.dialogs_py.SelectionDialog method), 142
winfo_vrootheight()  
  (robot.libraries.dialogs_py.InputDialog method), 130
winfo_vrootheight()  
  (robot.libraries.dialogs_py.MessageDialog method), 116
winfo_vrootheight()  
  (robot.libraries.dialogs_py.MultipleSelectionDialog method), 156
winfo_vrootheight()  
  (robot.libraries.dialogs_py.PassFailDialog method), 169
winfo_vrootheight()  
  (robot.libraries.dialogs_py.SelectionDialog method), 143
winfo_vrootwidth()  
  (robot.libraries.dialogs_py.InputDialog method), 130
winfo_vrootwidth()  
  (robot.libraries.dialogs_py.MessageDialog method), 117
winfo_vrootwidth()  
  (robot.libraries.dialogs_py.MultipleSelectionDialog method), 156
winfo_vrootwidth()  
  (robot.libraries.dialogs_py.PassFailDialog method), 169
winfo_vrootwidth()  
  (robot.libraries.dialogs_py.SelectionDialog method), 143
winfo_vrootx()  
  (robot.libraries.dialogs_py.InputDialog method), 130
winfo_vrootx()  
  (robot.libraries.dialogs_py.MessageDialog method), 117
winfo_vrootx()  
  (robot.libraries.dialogs_py.MultipleSelectionDialog method), 156
winfo_vrootx()  
  (robot.libraries.dialogs_py.PassFailDialog method), 169
winfo_vrootx()  
  (robot.libraries.dialogs_py.SelectionDialog method), 143
winfo_vrooty()  
  (robot.libraries.dialogs_py.InputDialog method), 130
winfo_vrooty()  
  (robot.libraries.dialogs_py.MessageDialog method), 117
winfo_vrooty()  
  (robot.libraries.dialogs_py.MultipleSelectionDialog method), 156
winfo_vrooty()  
  (robot.libraries.dialogs_py.PassFailDialog method), 169
winfo_vrooty()  
  (robot.libraries.dialogs_py.SelectionDialog method), 143
winfo_width()  
  (robot.libraries.dialogs_py.MessageDialog method), 117
winfo_width()  
  (robot.libraries.dialogs_py.MultipleSelectionDialog method), 156
winfo_width()  
  (robot.libraries.dialogs_py.PassFailDialog method), 169
winfo_width()  
  (robot.libraries.dialogs_py.SelectionDialog method), 143
winfo_width()  
  (robot.libraries.dialogs_py.InputDialog method), 130
with_metaclass()  
  (in module robot.utils.compat), 314
WITH_NAME  
  (robot.parsing.lexer.tokens.EOS attribute), 214
WITH_NAME  
  (robot.parsing.lexer.tokens.Token attribute), 212
withdraw()  
  (robot.libraries.dialogs_py.InputDialog method), 130
withdraw()  
  (robot.libraries.dialogs_py.MessageDialog method), 117
withdraw()  
  (robot.libraries.dialogs_py.MultipleSelectionDialog method), 156
withdraw()  
  (robot.libraries.dialogs_py.PassFailDialog method), 169
withdraw()  
  (robot.libraries.dialogs_py.SelectionDialog method), 143
wm_aspect()  
  (robot.libraries.dialogs_py.InputDialog method), 130
wm_aspect()  
  (robot.libraries.dialogs_py.MessageDialog method), 117
wm_aspect()  
  (robot.libraries.dialogs_py.MultipleSelectionDialog method), 156
wm_aspect()  
  (robot.libraries.dialogs_py.PassFailDialog method), 169
wm_aspect()  
  (robot.libraries.dialogs_py.SelectionDialog method), 143

Index

451
Robot Framework Documentation, Release 3.2b3.dev1

wm_deiconify() (robot.libraries.dialogs_py.SelectionDialog method), 143
wm_deiconify() (robot.libraries.dialogs_py.PassFailDialog method), 143
wm_deiconify() (robot.libraries.dialogs_py.MessageDialog method), 131
wm_deiconify() (robot.libraries.dialogs_py.MultipleSelectionDialog method), 157
wm_deiconify() (robot.libraries.dialogs_py.SelectionDialog method), 131
wm_deiconify() (robot.libraries.dialogs_py.PassFailDialog method), 157
wm_deiconify() (robot.libraries.dialogs_py.PassFailDialog method), 170
wm_deiconify() (robot.libraries.dialogs_py.PassFailDialog method), 169
wm_deiconify() (robot.libraries.dialogs_py.PassFailDialog method), 169
wm_deiconify() (robot.libraries.dialogs_py.MessageDialog method), 131
wm_deiconify() (robot.libraries.dialogs_py.MessageDialog method), 144
wm_deiconify() (robot.libraries.dialogs_py.MessageDialog method), 131
wm_deiconify() (robot.libraries.dialogs_py.SelectionDialog method), 131
wm_deiconify() (robot.libraries.dialogs_py.SelectionDialog method), 143
wm_deiconify() (robot.libraries.dialogs_py.SelectionDialog method), 131
wm_deiconify() (robot.libraries.dialogs_py.SelectionDialog method), 131
wm_deiconify() (robot.libraries.dialogs_py.SelectionDialog method), 131
wm_deiconify() (robot.libraries.dialogs_py.SelectionDialog method), 131
wm_deiconify() (robot.libraries.dialogs_py.SelectionDialog method), 131
wm_deiconify() (robot.libraries.dialogs_py.SelectionDialog method), 131
wm_deiconify() (robot.libraries.dialogs_py.SelectionDialog method), 131
wm_deiconify() (robot.libraries.dialogs_py.SelectionDialog method), 131
wm_deiconify() (robot.libraries.dialogs_py.SelectionDialog method), 131
wm_iconbitmap() (robot.libraries.dialogs_py.MessageDialog method), 118
wm_iconwindow() (robot.libraries.dialogs_py.MessageDialog method), 118
wm_iconbitmap() (robot.libraries.dialogs_py.MultilineSelectionDialog method), 157
wm_iconwindow() (robot.libraries.dialogs_py.MultilineSelectionDialog method), 157
wm_iconbitmap() (robot.libraries.dialogs_py.PassFailDialog method), 170
wm_iconwindow() (robot.libraries.dialogs_py.PassFailDialog method), 170
wm_iconbitmap() (robot.libraries.dialogs_py.SelectionDialog method), 144
wm_iconwindow() (robot.libraries.dialogs_py.SelectionDialog method), 144
wm_iconify() (robot.libraries.dialogs_py.InputDialog method), 131
wm_iconify() (robot.libraries.dialogs_py.MessageDialog method), 118
wm_iconify() (robot.libraries.dialogs_py.MultilineSelectionDialog method), 157
wm_iconify() (robot.libraries.dialogs_py.PassFailDialog method), 170
wm_iconify() (robot.libraries.dialogs_py.SelectionDialog method), 144
wm_iconmask() (robot.libraries.dialogs_py.InputDialog method), 131
wm_iconmask() (robot.libraries.dialogs_py.MessageDialog method), 118
wm_iconmask() (robot.libraries.dialogs_py.MultilineSelectionDialog method), 157
wm_iconmask() (robot.libraries.dialogs_py.PassFailDialog method), 170
wm_iconmask() (robot.libraries.dialogs_py.SelectionDialog method), 144
wm_iconname() (robot.libraries.dialogs_py.InputDialog method), 131
wm_iconname() (robot.libraries.dialogs_py.MessageDialog method), 118
wm_iconname() (robot.libraries.dialogs_py.MultilineSelectionDialog method), 157
wm_iconname() (robot.libraries.dialogs_py.PassFailDialog method), 170
wm_iconname() (robot.libraries.dialogs_py.SelectionDialog method), 144
wm_iconposition() (robot.libraries.dialogs_py.InputDialog method), 131
wm_iconposition() (robot.libraries.dialogs_py.MessageDialog method), 118
wm_iconposition() (robot.libraries.dialogs_py.MultilineSelectionDialog method), 157
wm_iconposition() (robot.libraries.dialogs_py.PassFailDialog method), 170
wm_iconposition() (robot.libraries.dialogs_py.SelectionDialog method), 144
wm_iconwindow() (robot.libraries.dialogs_py.InputDialog positionfrom())
write() (robot.output.output.Output method), 203
write() (robot.parsing.model.blocks.ModelWriter method), 217
write() (robot.reporting.jswriter.JsResultWriter method), 240
write() (robot.reporting.jswriter.SplitLogWriter method), 240
write() (robot.reporting.jswriter.SuiteWriter method), 240
write() (robot.reporting.logreportwriters.LogWriter method), 240
write() (robot.reporting.logreportwriters.ReportWriter method), 240
write() (robot.reporting.logreportwriters.RobotModelWriter method), 240
write() (robot.reporting.xunitwriter.XUnitWriter method), 243
write_bare() (robot.libraries.Telnet.TelnetConnection method), 91
write_control_character() (robot.libraries.Telnet.TelnetConnection method), 91
write_data() (robot.libdocpkg.htmlwriter.LibdocModelWriter method), 22
write_data() (robot.testdoc.TestdocModelWriter method), 345
write_json() (robot.htmldata.jsonwriter.JsonWriter method), 20
write_results() (robot.reporting.resultwriter.ResultWriter method), 242
write_until_expected_output() (robot.libraries.Telnet.TelnetConnection method), 91

X
XML (class in robot.libraries.XML), 95
xml_escape() (in module robot.utils.markuputils), 320
XmlElementHandler (class in robot.result.xmlelementhandlers), 271
XmlLogger (class in robot.output.xmllogger), 205
XmlRpcRemoteClient (class in robot.libraries.Remote), 77
XmlWriter (class in robot.libraries.xmlwriter), 321
xunit (robot.conf.settings.RebotSettings attribute), 19
xunit (robot.conf.settings.RobotSettings attribute), 19
xunit_skip_noncritical (robot.conf.settings.RebotSettings attribute), 19
xunit_skip_noncritical (robot.conf.settings.RobotSettings attribute), 19
XUnitFileWriter (class in robot.reporting.xunitwriter), 243

Y
YamlImporter (class in robot.variables.filesetter), 330
yellow() (robot.output.console.highlighting.AnsiHighlighter method), 196
yellow() (robot.output.console.highlighting.DosHighlighter method), 196
yellow() (robot.output.console.highlighting.NoHighlighting method), 196