

---

# **Robot Framework Documentation**

***Release 5.0.1***

**Robot Framework developers**

**May 16, 2022**



---

## Contents

---

<b>1</b>	<b>Entry points</b>	<b>3</b>
<b>2</b>	<b>Public API</b>	<b>5</b>
<b>3</b>	<b>All packages</b>	<b>7</b>
3.1	robot package . . . . .	7
<b>4</b>	<b>Indices</b>	<b>587</b>
	<b>Python Module Index</b>	<b>589</b>
	<b>Index</b>	<b>593</b>



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.



---

## Entry points

---

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

- `robot.run` entry point for executing tests.
- `robot.rebot` entry point for post-processing outputs (Rebot).
- `robot.libdoc` entry point for Libdoc tool.
- `robot.testdoc` entry point for Testdoc tool.

See [built-in tool documentation](#) for more details about Rebot, Libdoc, and Testdoc tools.





## CHAPTER 2

---

### Public API

---

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

Unless stated otherwise, the APIs exposed in this package are considered stable, and thus safe to use when building external tools on top of Robot Framework. Notice that all parsing APIs were rewritten in Robot Framework 3.2.

Currently exposed APIs are:

- `logger` module for libraries' logging purposes.
- `deco` module with decorators libraries can utilize.
- `exceptions` module containing exceptions that libraries can utilize for reporting failures and other events. These exceptions can be imported also directly via `robot.api` like `from robot.api import SkipExecution`.
- `parsing` module exposing the parsing APIs. This module is new in Robot Framework 4.0. Various parsing related functions and classes were exposed directly via `robot.api` already in Robot Framework 3.2, but they are effectively deprecated and will be removed in the future.
- `TestSuite` class for creating executable test suites programmatically and `TestSuiteBuilder` class for creating such suites based on existing test data on the file system.
- `SuiteVisitor` abstract class for processing testdata before execution. This can be used as a base for implementing a pre-run modifier that is taken into use with `--prerunmodifier` commandline option.
- `ExecutionResult()` factory method for reading execution results from XML output files and `ResultVisitor` abstract class to ease further processing the results. `ResultVisitor` can also be used as a base for pre-Rebot modifier that is taken into use with `--prerebotmodifier` commandline option.
- `ResultWriter` class for writing reports, logs, XML outputs, and XUnit files. Can write results based on XML outputs on the file system, as well as based on the result objects returned by the `ExecutionResult()` or an executed `TestSuite`.

All of the above names can be imported like:

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

See documentations of the individual APIs for more details.

---

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

---

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.

### 3.1 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.

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

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

---

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

---

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

Programmatic entry point for running tests.

#### Parameters

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

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

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

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

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

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

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

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

Example:

```
from robot import run

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

Equivalent command line usage:

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

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

Command line execution entry point for running tests.

#### Parameters

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

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

Example:

```

from robot import run_cli

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

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

```

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

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

Programmatic entry point for post-processing outputs.

#### Parameters

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

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

Examples:

```

from robot import rebot

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

```

Equivalent command line usage:

```

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

```

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

Command line execution entry point for post-processing outputs.

#### Parameters

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

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.

### 3.1.1 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. Notice that all parsing APIs were rewritten in Robot Framework 3.2.

Currently exposed APIs are:

- `logger` module for libraries' logging purposes.
- `deco` module with decorators libraries can utilize.
- `exceptions` module containing exceptions that libraries can utilize for reporting failures and other events. These exceptions can be imported also directly via `robot.api` like from `robot.api import SkipExecution`.
- `parsing` module exposing the parsing APIs. This module is new in Robot Framework 4.0. Various parsing related functions and classes were exposed directly via `robot.api` already in Robot Framework 3.2, but they are effectively deprecated and will be removed in the future.
- `TestSuite` class for creating executable test suites programmatically and `TestSuiteBuilder` class for creating such suites based on existing test data on the file system.
- `SuiteVisitor` abstract class for processing testdata before execution. This can be used as a base for implementing a pre-run modifier that is taken into use with `--prerunmodifier` commandline option.
- `ExecutionResult()` factory method for reading execution results from XML output files and `ResultVisitor` abstract class to ease further processing the results. `ResultVisitor` can also be used as a base for pre-Rebot modifier that is taken into use with `--prerebotmodifier` commandline option.
- `ResultWriter` class for writing reports, logs, XML outputs, and XUnit files. Can write results based on XML outputs on the file system, as well as based on the result objects returned by the `ExecutionResult()` or an executed `TestSuite`.

All of the above names can be imported like:

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

See documentations of the individual APIs for more details.

---

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

---

#### Submodules

##### robot.api.deco module

`robot.api.deco.not_keyword(func)`

Decorator to disable exposing functions or methods as keywords.

Examples:

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

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

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.

`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 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, converters=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`.

Arguments `scope`, `version`, `converters`, `doc_format` and `listener` set library's scope, version, converters, documentation format and listener by using class attributes `ROBOT_LIBRARY_SCOPE`, `ROBOT_LIBRARY_VERSION`, `ROBOT_LIBRARY_CONVERTERS`, `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:

```
@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. The `converters` argument is new in Robot Framework 5.0.

## robot.api.exceptions module

Exceptions that libraries can use for communicating failures and other events.

These exceptions can be imported also via the top level `robot.api` package like `from robot.api import SkipExecution`.

This module and all exceptions are new in Robot Framework 4.0.

**exception** `robot.api.exceptions.Failure` (*message*, *html=False*)

Bases: `AssertionError`

Report failed validation.

There is no practical difference in using this exception compared to using the standard `AssertionError`. The main benefits are HTML support and that the name of this exception is consistent with other exceptions in this module.

### Parameters

- **message** – Exception message.
- **html** – When `True`, message is considered to be HTML and not escaped.

**ROBOT\_SUPPRESS\_NAME** = `True`

### args

**with\_traceback()**

Exception.with\_traceback(tb) – set self.\_\_traceback\_\_ to tb and return self.

**exception** `robot.api.exceptions.ContinuableFailure` (*message*, *html=False*)

Bases: `robot.api.exceptions.Failure`

Report failed validation but allow continuing execution.



**Parameters**

- **message** – Exception message.
- **html** – When `True`, message is considered to be HTML and not escaped.

**ROBOT\_CONTINUE\_ON\_FAILURE = True**

**ROBOT\_SUPPRESS\_NAME = True**

**args**

**with\_traceback()**

Exception.with\_traceback(tb) – set self.\_\_traceback\_\_ to tb and return self.

**exception** `robot.api.exceptions.Error(message, html=False)`

Bases: `RuntimeError`

Report error in execution.

Failures related to the system not behaving as expected should typically be reported using the `Failure` exception or the standard `AssertionError`. This exception can be used, for example, if the keyword is used incorrectly.

There is no practical difference in using this exception compared to using the standard `RuntimeError`. The main benefits are HTML support and that the name of this exception is consistent with other exceptions in this module.

**Parameters**

- **message** – Exception message.
- **html** – When `True`, message is considered to be HTML and not escaped.

**ROBOT\_SUPPRESS\_NAME = True**

**args**

**with\_traceback()**

Exception.with\_traceback(tb) – set self.\_\_traceback\_\_ to tb and return self.

**exception** `robot.api.exceptions.FatalError(message, html=False)`

Bases: `robot.api.exceptions.Error`

Report error that stops the whole execution.

**Parameters**

- **message** – Exception message.
- **html** – When `True`, message is considered to be HTML and not escaped.

**ROBOT\_EXIT\_ON\_FAILURE = True**

**ROBOT\_SUPPRESS\_NAME = False**

**args**

**with\_traceback()**

Exception.with\_traceback(tb) – set self.\_\_traceback\_\_ to tb and return self.

**exception** `robot.api.exceptions.SkipExecution(message, html=False)`

Bases: `Exception`

Mark the executed test or task skipped.

**Parameters**

- **message** – Exception message.
- **html** – When `True`, message is considered to be HTML and not escaped.

```
ROBOT_SKIP_EXECUTION = True
```

```
ROBOT_SUPPRESS_NAME = True
```

```
args
```

```
with_traceback()
```

```
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.
```

## robot.api.logger module

Public logging API for test libraries.

This module provides a public API for writing messages to the log file and the console. Test libraries can use this API like:

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

instead of logging through the standard output like:

```
print('*INFO* My message')
```

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

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

## Log levels

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

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

## Logging HTML

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

## Example

```
from robot.api import logger

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

```
robot.api.logger.write(msg, level='INFO', html=False)
```

Writes the message to the log file using the given level.

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

```
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.api.parsing module

Public API for parsing, inspecting and modifying test data.

### Exposed API

The publicly exposed parsing entry points are the following:

- `get_tokens()`, `get_resource_tokens()`, and `get_init_tokens()` functions for *parsing data to tokens*.
- `Token` class that contains all token types as class attributes.
- `get_model()`, `get_resource_model()`, and `get_init_model()` functions for *parsing data to model* represented as an abstract syntax tree (AST).
- `Model` objects used by the AST model.
- `ModelVisitor` to ease *inspecting model* and *modifying data*.
- `ModelTransformer` for *adding and removing nodes*.

---

**Note:** This module is new in Robot Framework 4.0. In Robot Framework 3.2 functions for getting tokens and model as well as the `Token` class were exposed directly via the `robot.api` package, but other parts of the parsing API

were not publicly exposed. All code targeting Robot Framework 4.0 or newer should use this module because parsing related functions and classes will be removed from `robot.api` in the future.

---

**Note:** Parsing was totally rewritten in Robot Framework 3.2 and external tools using the parsing APIs need to be updated. Depending on the use case, it may be possible to use the higher level `TestSuiteBuilder()` instead.

---

### Parsing data to tokens

Data can be parsed to tokens by using `get_tokens()`, `get_resource_tokens()` or `get_init_tokens()` functions depending on whether the data represent a test case (or task) file, a resource file, or a suite initialization file. In practice the difference between these functions is what settings and sections are valid.

Typically the data is easier to inspect and modify by using the higher level model discussed in the next section, but in some cases having just the tokens can be enough. Tokens returned by the aforementioned functions are `Token` instances and they have the token type, value, and position easily available as their attributes. Tokens also have useful string representation used by the example below:

```
from robot.api.parsing import get_tokens

path = 'example.robot'

for token in get_tokens(path):
    print(repr(token))
```

If the `example.robot` used by the above example would contain

```
*** Test Cases ***
Example
    Keyword    argument

Second example
    Keyword    xxx

*** Keywords ***
Keyword
    [Arguments]    ${arg}
    Log            ${arg}
```

then the beginning of the output got when running the earlier code would look like this:

```
Token(TESTCASE_HEADER, '*** Test Cases ***', 1, 0)
Token(EOL, '\n', 1, 18)
Token(EOS, '', 1, 19)
Token(TESTCASE_NAME, 'Example', 2, 0)
Token(EOL, '\n', 2, 7)
Token(EOS, '', 2, 8)
Token(SEPARATOR, ' ', 3, 0)
Token(KEYWORD, 'Keyword', 3, 4)
Token(SEPARATOR, ' ', 3, 11)
Token(ARGUMENT, 'argument', 3, 15)
Token(EOL, '\n', 3, 23)
Token(EOS, '', 3, 24)
Token(EOL, '\n', 4, 0)
Token(EOS, '', 4, 1)
```

The output shows the token type, value, line number and column offset. When finding tokens by their type, the constants in the `Token` class such as `Token.TESTCASE_NAME` and `Token.EOL` should be used instead the values of these constants like `'TESTCASE NAME'` and `'EOL'`. These values have changed slightly in Robot Framework 4.0 and they may change in the future as well.

The `EOL` tokens denote end of a line and they include the newline character and possible trailing spaces. The `EOS` tokens denote end of a logical statement. Typically a single line forms a statement, but when the `...` syntax is used for continuation, a statement spans multiple lines. In special cases a single line can also contain multiple statements.

Errors caused by unrecognized data such as non-existing section or setting names are handled during the tokenizing phase. Such errors are reported using tokens that have `ERROR` type and the actual error message in their `error` attribute. Syntax errors such as empty `FOR` loops are only handled when building the higher level model discussed below.

See the documentation of `get_tokens()` for details about different ways how to specify the data to be parsed, how to control should all tokens or only data tokens be returned, and should variables in keyword arguments and elsewhere be tokenized or not.

## Parsing data to model

Data can be parsed to a higher level model by using `get_model()`, `get_resource_model()`, or `get_init_model()` functions depending on the type of the parsed file same way as when *parsing data to tokens*.

The model is represented as an abstract syntax tree (AST) implemented on top of Python's standard `ast.AST` class. To see how the model looks like, it is possible to use the `ast.dump()` function or the third-party `astpretty` module:

```
import ast
import astpretty
from robot.api.parsing import get_model

model = get_model('example.robot')
print(ast.dump(model, include_attributes=True))
print('-' * 72)
astpretty.pprint(model)
```

Running this code with the `example.robot` file from the previous section would produce so much output that it is not included here. If you are going to work with Robot Framework's AST, you are recommended to try that on your own.

## Model objects

The model is build from nodes that are based `ast.AST` and further categorized to blocks and statements. Blocks can contain other blocks and statements as child nodes whereas statements only have tokens containing the actual data as `Token` instances. Both statements and blocks expose their position information via `lineno`, `col_offset`, `end_lineno` and `end_col_offset` attributes and some nodes have also other special attributes available.

Blocks:

- `File` (the root of the model)
- `SettingSection`
- `VariableSection`
- `TestCaseSection`
- `KeywordSection`

- *CommentSection*
- *TestCase*
- *Keyword*
- *If*
- *Try*
- *For*
- *While*

### Statements:

- *SectionHeader*
- *LibraryImport*
- *ResourceImport*
- *VariablesImport*
- *Documentation*
- *Metadata*
- *ForceTags*
- *DefaultTags*
- *SuiteSetup*
- *SuiteTeardown*
- *TestSetup*
- *TestTeardown*
- *TestTemplate*
- *TestTimeout*
- *Variable*
- *TestCaseName*
- *KeywordName*
- *Setup*
- *Teardown*
- *Tags*
- *Template*
- *Timeout*
- *Arguments*
- *Return*
- *KeywordCall*
- *TemplateArguments*
- *IfHeader*
- *InlineIfHeader*

- *ElseIfHeader*
- *ElseHeader*
- *TryHeader*
- *ExceptHeader*
- *FinallyHeader*
- *ForHeader*
- *WhileHeader*
- *End*
- *ReturnStatement*
- *Break*
- *Continue*
- *Comment*
- *Error*
- *EmptyLine*

## Inspecting model

The easiest way to inspect what data a model contains is implementing *ModelVisitor* and creating *visit\_NodeName* to visit nodes with name *NodeName* as needed. The following example illustrates how to find what tests a certain test case file contains:

```
from robot.api.parsing import get_model, ModelVisitor

class TestNamePrinter(ModelVisitor):

    def visit_File(self, node):
        print(f"File '{node.source}' has following tests:")
        # Call `generic_visit` to visit also child nodes.
        self.generic_visit(node)

    def visit_TestCaseName(self, node):
        print(f"- {node.name} (on line {node.lineno})")

model = get_model('example.robot')
printer = TestNamePrinter()
printer.visit(model)
```

When the above code is run using the earlier `example.robot`, the output is this:

```
File 'example.robot' has following tests:
- Example (on line 2)
- Second example (on line 5)
```

## Handling errors in model

All nodes in the model have `errors` attribute that contains possible errors the node has. These errors include syntax errors such as empty FOR loops or IF without a condition as well as errors caused by unrecognized data such as non-existing section or setting names.

Unrecognized data is handled already during the *tokenizing* phase. In the model such data is represented as *Error* nodes and their `errors` attribute contain error information got from the underlying `ERROR` tokens. Syntax errors do not create *Error* nodes, but instead the model has normal nodes such as *If* with errors in their `errors` attribute.

A simple way to go through the model and see are there errors is using the *ModelVisitor* discussed in the previous section:

```
class ErrorReporter(ModelVisitor):

    # Implement `generic_visit` to visit all nodes.
    def generic_visit(self, node):
        if node.errors:
            print(f'Error on line {node.lineno}:')
            for error in node.errors:
                print(f'- {error}')
        ModelVisitor.generic_visit(self, node)
```

## Modifying data

Existing data the model contains can be modified simply by modifying values of the underlying tokens. If changes need to be saved, that is as easy as calling the *save()* method of the root model object. When just modifying token values, it is possible to still use *ModelVisitor* discussed in the above section. The next section discusses adding or removing nodes and then *ModelTransformer* should be used instead.

Modifications to tokens obviously require finding the tokens to be modified. The first step is finding nodes containing the tokens by implementing needed *visit\_NodeName* methods. Then the exact token or tokens can be found using nodes' *get\_token()* or *get\_tokens()* methods. If only token values are needed, *get\_value()* or *get\_values()* can be used as a shortcut. First finding nodes and then the right tokens is illustrated by this keyword renaming example:

```
from robot.api.parsing import get_model, ModelVisitor, Token

class KeywordRenamer(ModelVisitor):

    def __init__(self, old_name, new_name):
        self.old_name = self.normalize(old_name)
        self.new_name = new_name

    def normalize(self, name):
        return name.lower().replace(' ', '_').replace('-', '_')

    def visit_KeywordName(self, node):
        '''Rename keyword definitions.'''
        if self.normalize(node.name) == self.old_name:
            token = node.get_token(Token.KEYWORD_NAME)
            token.value = self.new_name

    def visit_KeywordCall(self, node):
        '''Rename keyword usages.'''
```

(continues on next page)



(continued from previous page)

```

    if self.normalize(node.keyword) == self.old_name:
        token = node.get_token(Token.KEYWORD)
        token.value = self.new_name

model = get_model('example.robot')
renamer = KeywordRenamer('Keyword', 'New Name')
renamer.visit(model)
model.save()

```

If you run the above example using the earlier `example.robot`, you can see that the `Keyword` keyword has been renamed to `New Name`. Notice that a real keyword renamer needed to take into account also keywords used with setups, teardowns and templates.

When token values are changed, column offset of the other tokens on same line are likely to be wrong. This does not affect saving the model or other typical usages, but if it is a problem then the caller needs to update offsets separately.

### Adding and removing nodes

Bigger changes to the model are somewhat more complicated than just modifying existing token values. When doing this kind of changes, *ModelTransformer* should be used instead of *ModelVisitor* that was discussed in the previous sections.

Removing nodes is relative easy and is accomplished by returning `None` from `visit_NodeName` methods. Remember to return the original node, or possibly a replacement node, from all of these methods when you do not want a node to be removed.

Adding nodes requires constructing needed *Model objects* and adding them to the model. The following example demonstrates both removing and adding nodes. If you run it against the earlier `example.robot`, you see that the first test gets a new keyword, the second test is removed, and settings section with documentation is added.

```

from robot.api.parsing import (
    get_model, Documentation, EmptyLine, KeywordCall,
    ModelTransformer, SettingSection, SectionHeader, Token
)

class TestModifier(ModelTransformer):

    def visit_TestCase(self, node):
        # The matched `TestCase` node is a block with `header` and
        # `body` attributes. `header` is a statement with familiar
        # `get_token` and `get_value` methods for getting certain
        # tokens or their value.
        name = node.header.get_value(Token.TESTCASE_NAME)
        # Returning `None` drops the node altogether i.e. removes
        # this test.
        if name == 'Second example':
            return None
        # Construct new keyword call statement from tokens. See `visit_File`
        # below for an example creating statements using `from_params`.
        new_keyword = KeywordCall([
            Token(Token.SEPARATOR, ' '),
            Token(Token.KEYWORD, 'New Keyword'),
            Token(Token.SEPARATOR, ' '),

```

(continues on next page)

(continued from previous page)

```

        Token(Token.ARGUMENT, 'xxx'),
        Token(Token.EOL)
    ])
    # Add the keyword call to test as the second item.
    node.body.insert(1, new_keyword)
    # No need to call `generic_visit` because we are not
    # modifying child nodes. The node itself must to be
    # returned to avoid dropping it.
    return node

def visit_File(self, node):
    # Create settings section with documentation. Needed header and body
    # statements are created using `from_params` method. This is typically
    # more convenient than creating statements based on tokens like above.
    settings = SettingSection(
        header=SectionHeader.from_params(Token.SETTING_HEADER),
        body=[
            Documentation.from_params('This is a really\npowerful API!'),
            EmptyLine.from_params()
        ]
    )
    # Add settings to the beginning of the file.
    node.sections.insert(0, settings)
    # Call `generic_visit` to visit also child nodes.
    return self.generic_visit(node)

model = get_model('example.robot')
TestModifier().visit(model)
model.save('modified.robot')

```

## Executing model

It is possible to convert a parsed and possibly modified model into an executable *TestSuite* structure by using its *from\_model()* class method. In this case the *get\_model()* function should be given the *curdir* argument to get possible `${CURDIR}` variable resolved correctly.

```

from robot.api import TestSuite
from robot.api.parsing import get_model

model = get_model('example.robot', curdir='/home/robot/example')
# modify model as needed
suite = TestSuite.from_model(model)
suite.run()

```

For more details about executing the created *TestSuite* object, see the documentation of its *run()* method. Notice also that if you do not need to modify the parsed model, it is easier to get the executable suite by using the *from\_file\_system()* class method.

## 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 tests.

Can be overridden to allow modifying the passed in *test* without calling *start\_test()* or *end\_test()* nor visiting the body of the test.

**visit\_keyword** (*kw*)

Implements traversing through keywords.

Can be overridden to allow modifying the passed in *kw* without calling *start\_keyword()* or *end\_keyword()* nor visiting the body of the keyword

**end\_body\_item** (*item*)

Called, by default, when keywords, messages or control structures end.

More specific *end\_keyword()*, *end\_message()*, *:meth:'end\_for'*, etc. can be implemented to visit only keywords, messages or specific control structures.

Default implementation does nothing.

**end\_break** (*break\_*)

Called when a BREAK element ends.

By default, calls *end\_body\_item()* which, by default, does nothing.

**end\_continue** (*continue\_*)

Called when a CONTINUE element ends.

By default, calls *end\_body\_item()* which, by default, does nothing.

**end\_for** (*for\_*)

Called when a FOR loop ends.

By default, calls *end\_body\_item()* which, by default, does nothing.

**end\_for\_iteration** (*iteration*)

Called when a FOR loop iteration ends.

By default, calls *end\_body\_item()* which, by default, does nothing.

**end\_if** (*if\_*)

Called when an IF/ELSE structure ends.

By default, calls *end\_body\_item()* which, by default, does nothing.

**end\_if\_branch** (*branch*)

Called when an IF/ELSE branch ends.

By default, calls *end\_body\_item()* which, by default, does nothing.

**end\_keyword** (*keyword*)

Called when a keyword ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_message** (*msg*)

Called when a message ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_return** (*return\_*)

Called when a RETURN element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_suite** (*suite*)

Called when a suite ends. Default implementation does nothing.

**end\_test** (*test*)

Called when a test ends. Default implementation does nothing.

**end\_try** (*try\_*)

Called when a TRY/EXCEPT structure ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_try\_branch** (*branch*)

Called when TRY, EXCEPT, ELSE and FINALLY branches end.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_while** (*while\_*)

Called when a WHILE loop ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_while\_iteration** (*iteration*)

Called when a WHILE loop iteration ends.

By default, calls `end_body_item()` which, by default, does nothing.

**start\_body\_item** (*item*)

Called, by default, when keywords, messages or control structures start.

More specific `start_keyword()`, `start_message()`, `:meth:'start_for`, etc. can be implemented to visit only keywords, messages or specific control structures.

Can return explicit `False` to stop visiting. Default implementation does nothing.

**start\_break** (*break\_*)

Called when a BREAK element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_continue** (*continue\_*)

Called when a CONTINUE element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_for** (*for\_*)

Called when a FOR loop starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_for\_iteration** (*iteration*)

Called when a FOR loop iteration starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_if** (*if\_*)

Called when an IF/ELSE structure starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_if\_branch** (*branch*)

Called when an IF/ELSE branch starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_keyword** (*keyword*)

Called when a keyword starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_message** (*msg*)

Called when a message starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_return** (*return\_*)

Called when a RETURN element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_suite** (*suite*)

Called when a suite starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

**start\_test** (*test*)

Called when a test starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

**start\_try** (*try\_*)

Called when a TRY/EXCEPT structure starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_try\_branch** (*branch*)

Called when TRY, EXCEPT, ELSE or FINALLY branches start.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_while** (*while\_*)

Called when a WHILE loop starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_while\_iteration** (*iteration*)

Called when a WHILE loop iteration starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**visit\_break** (*break\_*)

Visits BREAK elements.

**visit\_continue** (*continue\_*)

Visits CONTINUE elements.

**visit\_for** (*for\_*)

Implements traversing through FOR loops.

Can be overridden to allow modifying the passed in `for_` without calling `start_for()` or `end_for()` nor visiting body.

**visit\_for\_iteration** (*iteration*)

Implements traversing through single FOR loop iteration.

This is only used with the result side model because on the running side there are no iterations.

Can be overridden to allow modifying the passed in `iteration` without calling `start_for_iteration()` or `end_for_iteration()` nor visiting body.

**visit\_if** (*if\_*)

Implements traversing through IF/ELSE structures.

Notice that `if_` does not have any data directly. Actual IF/ELSE branches are in its `body` and visited using `visit_if_branch()`.

Can be overridden to allow modifying the passed in `if_` without calling `start_if()` or `end_if()` nor visiting branches.

**visit\_if\_branch** (*branch*)

Implements traversing through single IF/ELSE branch.

Can be overridden to allow modifying the passed in `branch` without calling `start_if_branch()` or `end_if_branch()` nor visiting body.

**visit\_message** (*msg*)

Implements visiting messages.

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

**visit\_return** (*return\_*)

Visits a RETURN elements.

**visit\_suite** (*suite*)

Implements traversing through suites.

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

**visit\_try** (*try\_*)

Implements traversing through TRY/EXCEPT structures.

This method is used with the TRY/EXCEPT root element. Actual TRY, EXCEPT, ELSE and FINALLY branches are visited separately using `visit_try_branch()`.

**visit\_try\_branch** (*branch*)

Visits individual TRY, EXCEPT, ELSE and FINALLY branches.

**visit\_while** (*while\_*)

Implements traversing through WHILE loops.

Can be overridden to allow modifying the passed in `while_` without calling `start_while()` or `end_while()` nor visiting body.

**visit\_while\_iteration** (*iteration*)

Implements traversing through single WHILE loop iteration.

This is only used with the result side model because on the running side there are no iterations.

Can be overridden to allow modifying the passed in `iteration` without calling `start_while_iteration()` or `end_while_iteration()` nor visiting body.

**class** `robot.conf.gatherfailed.GatherFailedSuites`

Bases: `robot.model.visitor.SuiteVisitor`

**start\_suite** (*suite*)

Called when a suite starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

**visit\_test** (*test*)

Implements traversing through tests.

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

**visit\_keyword** (*kw*)

Implements traversing through keywords.

Can be overridden to allow modifying the passed in `kw` without calling `start_keyword()` or `end_keyword()` nor visiting the body of the keyword

**end\_body\_item** (*item*)

Called, by default, when keywords, messages or control structures end.

More specific `end_keyword()`, `end_message()`, `:meth:'end_for'`, etc. can be implemented to visit only keywords, messages or specific control structures.

Default implementation does nothing.

**end\_break** (*break\_*)

Called when a BREAK element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_continue** (*continue\_*)

Called when a CONTINUE element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_for** (*for\_*)

Called when a FOR loop ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_for\_iteration** (*iteration*)

Called when a FOR loop iteration ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_if** (*if\_*)

Called when an IF/ELSE structure ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_if\_branch** (*branch*)

Called when an IF/ELSE branch ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_keyword** (*keyword*)

Called when a keyword ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_message** (*msg*)

Called when a message ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_return** (*return\_*)

Called when a RETURN element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_suite** (*suite*)

Called when a suite ends. Default implementation does nothing.

**end\_test** (*test*)

Called when a test ends. Default implementation does nothing.

**end\_try** (*try\_*)

Called when a TRY/EXCEPT structure ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_try\_branch** (*branch*)

Called when TRY, EXCEPT, ELSE and FINALLY branches end.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_while** (*while\_*)

Called when a WHILE loop ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_while\_iteration** (*iteration*)

Called when a WHILE loop iteration ends.

By default, calls `end_body_item()` which, by default, does nothing.

**start\_body\_item** (*item*)

Called, by default, when keywords, messages or control structures start.

More specific `start_keyword()`, `start_message()`, `:meth:'start_for`, etc. can be implemented to visit only keywords, messages or specific control structures.

Can return explicit `False` to stop visiting. Default implementation does nothing.



**start\_break** (*break\_*)

Called when a BREAK element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_continue** (*continue\_*)

Called when a CONTINUE element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_for** (*for\_*)

Called when a FOR loop starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_for\_iteration** (*iteration*)

Called when a FOR loop iteration starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_if** (*if\_*)

Called when an IF/ELSE structure starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_if\_branch** (*branch*)

Called when an IF/ELSE branch starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_keyword** (*keyword*)

Called when a keyword starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_message** (*msg*)

Called when a message starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_return** (*return\_*)

Called when a RETURN element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_test** (*test*)

Called when a test starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

**start\_try** (*try\_*)

Called when a TRY/EXCEPT structure starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_try\_branch** (*branch*)

Called when TRY, EXCEPT, ELSE or FINALLY branches start.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_while** (*while\_*)

Called when a WHILE loop starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_while\_iteration** (*iteration*)

Called when a WHILE loop iteration starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**visit\_break** (*break\_*)

Visits BREAK elements.

**visit\_continue** (*continue\_*)

Visits CONTINUE elements.

**visit\_for** (*for\_*)

Implements traversing through FOR loops.

Can be overridden to allow modifying the passed in `for_` without calling `start_for()` or `end_for()` nor visiting body.

**visit\_for\_iteration** (*iteration*)

Implements traversing through single FOR loop iteration.

This is only used with the result side model because on the running side there are no iterations.

Can be overridden to allow modifying the passed in `iteration` without calling `start_for_iteration()` or `end_for_iteration()` nor visiting body.

**visit\_if** (*if\_*)

Implements traversing through IF/ELSE structures.

Notice that `if_` does not have any data directly. Actual IF/ELSE branches are in its `body` and visited using `visit_if_branch()`.

Can be overridden to allow modifying the passed in `if_` without calling `start_if()` or `end_if()` nor visiting branches.

**visit\_if\_branch** (*branch*)

Implements traversing through single IF/ELSE branch.

Can be overridden to allow modifying the passed in `branch` without calling `start_if_branch()` or `end_if_branch()` nor visiting body.

**visit\_message** (*msg*)

Implements visiting messages.

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

**visit\_return**(*return\_*)  
Visits a RETURN elements.

**visit\_suite**(*suite*)  
Implements traversing through suites.

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

**visit\_try**(*try\_*)  
Implements traversing through TRY/EXCEPT structures.  
  
This method is used with the TRY/EXCEPT root element. Actual TRY, EXCEPT, ELSE and FINALLY branches are visited separately using `visit_try_branch()`.

**visit\_try\_branch**(*branch*)  
Visits individual TRY, EXCEPT, ELSE and FINALLY branches.

**visit\_while**(*while\_*)  
Implements traversing through WHILE loops.

Can be overridden to allow modifying the passed in `while_` without calling `start_while()` or `end_while()` nor visiting body.

**visit\_while\_iteration**(*iteration*)  
Implements traversing through single WHILE loop iteration.

This is only used with the result side model because on the running side there are no iterations.

Can be overridden to allow modifying the passed in `iteration` without calling `start_while_iteration()` or `end_while_iteration()` nor visiting body.

`robot.conf.gatherfailed.gather_failed_tests(output, empty_suite_ok=False)`

`robot.conf.gatherfailed.gather_failed_suites(output, empty_suite_ok=False)`

## 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**

**suite\_names**

**test\_names**

**randomize\_seed**

**randomize\_suites**

**randomize\_tests**

**dry\_run**

**exit\_on\_failure**

```
exit_on_error
skip
skipped_tags
skip_on_failure
skip_teardown_on_exit
console_output_config
console_type
console_width
console_markers
max_error_lines
max_assign_length
pre_run_modifiers
run_empty_suite
variables
variable_files
extension
console_colors
exclude
flatten_keywords
include
log
log_level
output
output_directory
pre_rebot_modifiers
pythonpath
remove_keywords
report
rpa
split_log
statistics_config
status_rc
xunit

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`  
`exclude`  
`flatten_keywords`  
`include`  
`log`  
`log_level`  
`output`  
`output_directory`  
`pre_rebot_modifiers`  
`process_empty_suite`  
`pythonpath`  
`remove_keywords`  
`report`  
`rpa`  
`split_log`  
`statistics_config`  
`status_rc`  
`suite_names`  
`test_names`  
`xunit`  
`expand_keywords`

### robot.htmldata package

Package for writing output files in HTML format.

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

### Submodules

#### robot.htmldata.htmlfilewriter module

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

```
class robot.htmldata.htmlfilewriter.ModelWriter
    Bases: robot.htmldata.htmlfilewriter._Writer

    handles (line)

    write (line)

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

    handles (line)

    write (line)

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

    write (line)

    handles (line)

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

    write (line)

    handles (line)

class robot.htmldata.htmlfilewriter.CssFileWriter (html_writer, base_dir)
    Bases: robot.htmldata.htmlfilewriter._InliningWriter

    write (line)

    handles (line)
```

### robot.htmldata.jsonwriter module

```
class robot.htmldata.jsonwriter.JsonWriter (output, separator="")
    Bases: object

    write_json (prefix, data, postfix=';\n', mapping=None, separator=True)

    write (string, postfix=';\n', separator=True)

class robot.htmldata.jsonwriter.JsonDumper (output)
    Bases: object

    dump (data, mapping=None)

class robot.htmldata.jsonwriter.StringDumper (jsondumper)
    Bases: robot.htmldata.jsonwriter._Dumper

    dump (data, mapping)

    handles (data, mapping)

class robot.htmldata.jsonwriter.IntegerDumper (jsondumper)
    Bases: robot.htmldata.jsonwriter._Dumper

    dump (data, mapping)

    handles (data, mapping)

class robot.htmldata.jsonwriter.DictDumper (jsondumper)
    Bases: robot.htmldata.jsonwriter._Dumper
```

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

### robot.htmldata.template module

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

### robot.libdocpkg package

Implements the *Libdoc* tool.

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

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

### Submodules

#### robot.libdocpkg.builder module

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

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

Create a documentation builder for the specified library or resource.

The argument can be a path to a library, a resource file or to a spec file generated by Libdoc earlier. If the argument does not point to an existing file, it is expected to be the name of the library to be imported. If a resource file is to be imported from PYTHONPATH, then *ResourceDocBuilder* must be used explicitly instead.

**robot.libdocpkg.consoleviewer module**

```
class robot.libdocpkg.consoleviewer.ConsoleViewer (libdoc)
    Bases: object

    classmethod handles (command)

    classmethod validate_command (command, args)

    view (command, *args)

    list (*patterns)

    show (*names)

    version ()

class robot.libdocpkg.consoleviewer.KeywordMatcher (libdoc)
    Bases: object

    search (patterns)
```

**robot.libdocpkg.datatypes module**

```
class robot.libdocpkg.datatypes.TypeDoc (type, name, doc, accepts=(), usages=None, members=None, items=None)
    Bases: robot.utils.sortable.Sortable

    ENUM = 'Enum'

    TYPED_DICT = 'TypedDict'

    CUSTOM = 'Custom'

    STANDARD = 'Standard'

    classmethod for_type (type, converters)

    classmethod for_enum (enum)

    classmethod for_typed_dict (typed_dict)

    to_dictionary (legacy=False)

class robot.libdocpkg.datatypes.TypedDictItem (key, type, required=None)
    Bases: object

    to_dictionary ()

class robot.libdocpkg.datatypes.EnumMember (name, value)
    Bases: object

    to_dictionary ()
```

**robot.libdocpkg.htmlutils module**

```
class robot.libdocpkg.htmlutils.DocFormatter (keywords, type_info, introduction, doc_format='ROBOT')
    Bases: object

    html (doc, intro=False)
```



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

class robot.libdocpkg.htmlutils.HtmlToText
    Bases: object

    html_tags = {'b': '*', 'code': '`', 'div.*?': '', 'em': '_', 'i': '_', 'strong': ''}
    html_chars = {'&': '&', '&apos;': "'", '&gt;': '>', '&lt;': '<', '&quot;': '"'}
    get_shortcode_from_html(doc)
    html_to_plain_text(doc)
```

### robot.libdocpkg.htmlwriter module

```
class robot.libdocpkg.htmlwriter.LibdocHtmlWriter
    Bases: object

    write(libdoc, output)

class robot.libdocpkg.htmlwriter.LibdocModelWriter(output, libdoc)
    Bases: robot.htmldata.htmlfilewriter.ModelWriter

    write(line)
    handles(line)
```

### robot.libdocpkg.jsonbuilder module

```
class robot.libdocpkg.jsonbuilder.JsonDocBuilder
    Bases: object

    build(path)
    build_from_dict(spec)
```

### robot.libdocpkg.jsonwriter module

```
class robot.libdocpkg.jsonwriter.LibdocJsonWriter
    Bases: object

    write(libdoc, outfile)
```

### robot.libdocpkg.model module

```
class robot.libdocpkg.model.LibraryDoc(name="", doc="", version="", type='LIBRARY',
                                       scope='TEST', doc_format='ROBOT',
                                       source=None, lineno=-1)

    Bases: object

    doc
    doc_format
    inits
    keywords
```

```
type_docs
all_tags
save (output=None, format='HTML')
convert_docs_to_html()
to_dictionary()
to_json(indent=None)
class robot.libdocpkg.model.KeywordDoc (name="", args=None, doc="", shortdoc="", tags=(),
                                         source=None, lineno=-1, parent=None)
    Bases: robot.utils.sortable.Sortable
    shortdoc
    deprecated
    to_dictionary()
```

#### robot.libdocpkg.output module

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

#### robot.libdocpkg.robotbuilder module

```
class robot.libdocpkg.robotbuilder.LibraryDocBuilder
    Bases: object
    build(library)
class robot.libdocpkg.robotbuilder.ResourceDocBuilder
    Bases: object
    build(path)
class robot.libdocpkg.robotbuilder.KeywordDocBuilder (resource=False)
    Bases: object
    build_keywords(lib)
    build_keyword(kw)
```

#### robot.libdocpkg.standardtypes module

#### robot.libdocpkg.writer module

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

#### robot.libdocpkg.xmlbuilder module

```
class robot.libdocpkg.xmlbuilder.XmlDocBuilder
    Bases: object
    build(path)
```

## robot.libdocpkg.xmlwriter module

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

## robot.libraries package

Package hosting Robot Framework standard test libraries.

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

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

## Submodules

### robot.libraries.BuiltIn module

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

```
class robot.libraries.BuiltIn.BuiltIn
    Bases: robot.libraries.BuiltIn._Verify, robot.libraries.BuiltIn._Converter,
           robot.libraries.BuiltIn._Variables, robot.libraries.BuiltIn._RunKeyword,
           robot.libraries.BuiltIn._Control, robot.libraries.BuiltIn._Misc
```

An always available standard library with often needed keywords.

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

== Table of contents ==

%TOC%

= 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.

= Using variables with keywords creating or accessing variables =

This library has special keywords *Set Global Variable*, *Set Suite Variable*, *Set Test Variable* and *Set Local Variable* for creating variables in different scopes. These keywords take the variable name and its value as arguments. The name can be given using the normal *\${variable}* syntax or in escaped format either like *\$variable* or *\\${variable}*. For example, these are typically equivalent and create new suite level variable *\${name}* with value *value*:

A problem with using the normal *\${variable}* syntax is that these keywords cannot easily know is the idea to create a variable with exactly that name or does that variable actually contain the name of the variable to create. If the variable does not initially exist, it will always be created. If it exists and its value is a variable

name either in the normal or in the escaped syntax, variable with `_that_` name is created instead. For example, if `${name}` variable would exist and contain value `$example`, these examples would create different variables:

Because the behavior when using the normal `${variable}` syntax depends on the possible existing value of the variable, it is *highly recommended to use the escaped “`$variable`” or “``${variable}`” format instead.*

This same problem occurs also with special keywords for accessing variables *Get Variable Value*, *Variable Should Exist* and *Variable Should Not Exist*.

= 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> 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 in special cases where the automatic module import does not work such as when using nested modules like `rootmod.submod` or list comprehensions. See the documentation of the *Evaluate* keyword for more details.

== Variables in expressions ==

When a variable is used in the expression 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 constraints.

Notice that instead of creating complicated expressions, it is often better to move the logic into a 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#truthrules> as in Python].

True examples:

False examples:

= Pattern matching =

Many keywords accept 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\]](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.

== Regular expressions ==

Some keywords, for example *Should Match Regexp*, support [\[http://en.wikipedia.org/wiki/Regular\\_expression#regular\\_expressions\]](http://en.wikipedia.org/wiki/Regular_expression#regular_expressions) that are more powerful but also more complicated than glob patterns. The regular expression support is implemented using Python's [\[http://docs.python.org/library/re.html#re-module\]](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 [\[http://en.wikipedia.org/wiki/Diff\\_utility#Unified\\_format#unified\\_diff\\_format\]](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 the 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 `ä` (`\u0061`) and the Cyrillic (`\u0430`). Error messages like `ä != ä` are not very helpful.
- Some Unicode characters can be represented using [\[https://en.wikipedia.org/wiki/Unicode\\_equivalence#different\\_forms\]](https://en.wikipedia.org/wiki/Unicode_equivalence#different_forms). For example, `ä` can be represented either as a single code point `\u00e4` or using two combined code points `\u0061` and `\u0308`. Such forms are considered canonically equivalent, but strings containing them are not considered equal when compared in Python. Error messages like `ä != ä` are not that helpful either.
- Containers such as lists and dictionaries are formatted into a single line making it hard to see individual items they contain.

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

== str ==

Use the human-readable string representation. Equivalent to using `str()` in Python. 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 this mode bigger lists, dictionaries and other containers are pretty-printed so that there is one item per row.

`== ascii ==`

Same as using `ascii()` in Python. Similar to using `repr` explained above but with the following differences:

- 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.
- Containers are not pretty-printed.

**ROBOT\_LIBRARY\_SCOPE** = 'GLOBAL'

**ROBOT\_LIBRARY\_VERSION** = '5.0.1'

**call\_method** (*object*, *method\_name*, *\*args*, *\*\*kwargs*)

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

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

Possible equal signs in arguments must be escaped with a backslash like `\=`.

**catenate** (*\*items*)

Catenates the given items together and returns the resulted string.

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

**comment** (*\*messages*)

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

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

**continue\_for\_loop** ()

Skips the current FOR loop iteration and continues from the next.

—

*NOTE:* Robot Framework 5.0 added support for native `CONTINUE` statement that is recommended over this keyword. In the examples below, `Continue For Loop` can simply be replaced with `CONTINUE`. In addition to that, native `IF` syntax (new in RF 4.0) or inline `IF` syntax (new in RF 5.0) can be used instead of `Run Keyword If`. For example, the first example below could be written like this instead:

This keyword will eventually be deprecated and removed.

—

Skips the remaining keywords in the current FOR loop iteration and continues from the next one. Starting from Robot Framework 5.0, this keyword can only be used inside a loop, not in a keyword used in a loop.

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.

—

*NOTE:* Robot Framework 5.0 added support for native `CONTINUE` statement and for inline `IF`, and that combination should be used instead of this keyword. For example, `Continue For Loop If` usage in the example below could be replaced with

This keyword will eventually be deprecated and removed.

—

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#truthtruth> value] using Python's `bool()` method.

**convert\_to\_bytes** (*input*, *input\_type='text'*)

Converts the given *input* to bytes according to the *input\_type*.

Valid input types are listed below:

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

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

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

**convert\_to\_hex** (*item*, *base=None*, *prefix=None*, *length=None*, *lowercase=False*)

Converts the given item to a hexadecimal string.

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

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

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

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

**convert\_to\_integer** (*item*, *base=None*)

Converts the given item to an integer number.

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

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

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

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

**convert\_to\_number** (*item*, *precision=None*)

Converts the given item to a floating point number.

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

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

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

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

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

**convert\_to\_octal** (*item*, *base=None*, *prefix=None*, *length=None*)

Converts the given item to an octal string.

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

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

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

**convert\_to\_string** (*item*)

Converts the given item to a Unicode string.

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

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



**create\_dictionary** (\*items)

Creates and returns a dictionary based on the given items.

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

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

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

**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.

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.

Starting from Robot Framework 3.2, modules used in the expression are imported automatically. There are, however, two cases where they need to be explicitly specified using the `modules` argument:

- When nested modules like `rootmod.submod` are implemented so that the root module does not automatically import sub modules. This is illustrated by the `selenium.webdriver` example below.
- When using a module in the expression part of a list comprehension. This is illustrated by the `json` example below.

*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.

—

*NOTE:* Robot Framework 5.0 added support for native `BREAK` statement that is recommended over this keyword. In the examples below, `Exit For Loop` can simply be replaced with `BREAK`. In addition to that, native `IF` syntax (new in RF 4.0) or inline `IF` syntax (new in RF 5.0) can be used instead of `Run Keyword If`. For example, the first example below could be written like this instead:

This keyword will eventually be deprecated and removed.

—

Exits the enclosing FOR loop and continues execution after it. Starting from Robot Framework 5.0, this keyword can only be used inside a loop, not in a keyword used in a loop.

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.

—

*NOTE:* Robot Framework 5.0 added support for native BREAK statement and for inline IF, and that combination should be used instead of this keyword. For example, *Exit For Loop If* usage in the example below could be replaced with

This keyword will eventually be deprecated and removed.

—

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** (*container, item*)

Returns and logs how many times `item` is found from `container`.

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 library.

This keyword makes it easy for libraries to interact with other 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 [\[http://en.wikipedia.org/wiki/Coordinated\\_Universal\\_Time|UTC\]](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 like `${name}` or in escaped format like `$name` or `\${name}`. For the reasons explained in the *Using variables with keywords creating or accessing variables* section, using the escaped format is recommended.

- `${x}` gets value of `${a}` if `${a}` exists and string `default` otherwise

- `${y}` gets value of `${a}` if `${a}` exists and value of `${b}` otherwise
- `${z}` is set to Python `None` if it does not exist previously

**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 section.

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#module-search-path>] 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#module-search-path>]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#module-search-path>]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='DEPRECATED', 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 `` 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*, *ascii*, *len*, and *type*. 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*.

The old way to control string representation was using the *repr* argument. This argument has been deprecated and *formatter=repr* should be used instead.

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.

Formatter options *type* and *log* are new in Robot Framework 5.0.

**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, format=""*)

Logs the given message to the console.

By default uses the standard output stream. Using the standard error stream is possible 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*).

By default adds no alignment formatting. The *format* argument allows, for example, alignment and customized padding of the log message. Please see the [<https://docs.python.org/3/library/string.html#formatspec>] for detailed alignment possibilities. This argument is new in Robot Framework 5.0.

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

**log\_variables** (*level='INFO'*)

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

**no\_operation** ()

Does absolutely nothing.

**pass\_execution** (*message, \*tags*)

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

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

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

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

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

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

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

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

**pass\_execution\_if** (*condition, message, \*tags*)

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

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

**regexp\_escape** (*\*patterns*)

Returns each argument string escaped for use as a regular expression.

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

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

**reload\_library** (*name\_or\_instance*)

Rechecks what keywords the specified library provides.

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

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

**remove\_tags** (*\*tags*)

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

Tags can be given exactly or using a pattern with `*`, `?` and `[chars]` acting as wildcards. See the *Glob patterns* section for more information.

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

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

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

**repeat\_keyword** (*repeat*, *name*, *\*args*)

Executes the specified keyword multiple times.

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

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

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

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

**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.

—

*NOTE:* Robot Framework 5.0 added support for native `RETURN` statement that is recommended over this keyword. In the examples below, `Return From Keyword` can simply be replaced with `RETURN`. In addition to that, native `IF` syntax (new in RF 4.0) or inline `IF` syntax (new in RF 5.0) can be used instead of `Run Keyword If`. For example, the first example below could be written like this instead:

This keyword will eventually be deprecated and removed.

—

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*, 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 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.

—

*NOTE:* Robot Framework 5.0 added support for native RETURN statement and for inline IF, and that combination should be used instead of this keyword. For example, *Return From Keyword* usage in the example below could be replaced with

This keyword will eventually be deprecated and removed.

—

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 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.

*NOTE:* Regular expression matching used to require only the beginning of the error to match the given pattern. That was changed in Robot Framework 5.0 and nowadays the pattern must match the error fully. To match only the beginning, add `.*` at the end of the pattern like `REGEXP: Start.*`.

*NOTE:* Robot Framework 5.0 introduced native TRY/EXCEPT functionality that is generally recommended for error handling. It supports same pattern matching syntax as 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.

*NOTE:* Robot Framework 5.0 introduced native TRY/EXCEPT functionality that is generally recommended for error handling.

#### **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\_and\_warn\_on\_failure** (*name*, \**args*)

Runs the specified keyword logs a warning if the keyword fails.

This keyword is similar to *Run Keyword And Ignore Error* but if the executed keyword fails, the error message is logged as a warning to make it more visible. Returns status and possible return value or error message exactly like *Run Keyword And Ignore Error* does.

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

New in Robot Framework 4.0.

#### **run\_keyword\_if** (*condition*, *name*, \**args*)

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

*NOTE:* Robot Framework 4.0 introduced built-in IF/ELSE support and using that is generally recommended over using this keyword.

The given *condition* is evaluated in Python as explained in the *Evaluating expressions* section, 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 value of the `${status}` variable.

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.

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* 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`.

**`run_keyword_if_all_tests_passed`** (*name*, *\*args*)

Runs the given keyword with the given arguments, if all tests passed.

This keyword can only be used in a suite teardown. Trying to use it anywhere else results in an error.

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

**`run_keyword_if_any_tests_failed`** (*name*, *\*args*)

Runs the given keyword with the given arguments, if one or more tests failed.

This keyword can only be used in a suite teardown. Trying to use it anywhere else results in an error.

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

**`run_keyword_if_test_failed`** (*name*, *\*args*)

Runs the given keyword with the given arguments, if the test failed.

This keyword can only be used in a test teardown. Trying to use it anywhere else results in an error.

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

**`run_keyword_if_test_passed`** (*name*, *\*args*)

Runs the given keyword with the given arguments, if the test passed.

This keyword can only be used in a test teardown. Trying to use it anywhere else results in an error.

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

**`run_keyword_if_timeout_occurred`** (*name*, *\*args*)

Runs the given keyword if either a test or a keyword timeout has occurred.

This keyword can only be used in a test teardown. Trying to use it anywhere else results in an error.

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

**`run_keyword_unless`** (*condition*, *name*, *\*args*)

*DEPRECATED since RF 5.0. Use Native IF/ELSE or 'Run Keyword If' instead.*

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.

**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 created Variables sections are overridden. Variables assigned locally based on keyword return values or by using *Set Suite Variable*, *Set Test Variable* or *Set Local 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 usage examples. See also the *Using variables with keywords creating or accessing variables* section for information why it is recommended to give the variable name in escaped format like \$name or \\${name} instead of the normal \${name}.

**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. For example, are equivalent with

The main use case for this keyword is creating local variables in libraries.

See *Set Suite Variable* for more information and usage examples. See also the *Using variables with keywords creating or accessing variables* section for information why it is recommended to give the variable name in escaped format like `$name` or `\${name}` instead of the normal `${name}`.

See also *Set Global Variable* and *Set Test Variable*.

**set\_log\_level** (*level*)

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

Messages below the level will not be 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 Variables section in the 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>` is 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 like `${NAME}` or in escaped format as `\${NAME}` or `$NAME`. For the reasons explained in the *Using variables with keywords creating or accessing variables* section, *using the escaped format is highly recommended*.

Variable value can be specified using the same syntax as when variables are created in the Variables section. Same way as in that section, it is possible to create scalar values, lists and dictionaries. The type is got from the variable name prefix `$`, `@` and `&`, respectively.

If a variable already exists within the new scope, its value will be overwritten. 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}`:

See also *Set Global Variable*, *Set Test Variable* and *Set Local Variable*.

#### **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.

#### **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. It is an error to call *Set Test Variable* outside the scope of a test (e.g. in a Suite Setup or Teardown).

See *Set Suite Variable* for more information and usage examples. See also the *Using variables with keywords creating or accessing variables* section for information why it is recommended to give the variable name in escaped format like `$name` or `\${name}` instead of the normal `${name}`.

When creating automated tasks, not tests, it is possible to use *Set Task Variable*. See also *Set Global Variable* and *Set Local Variable*.

**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', strip\_spaces=False, collapse\_spaces=False)

Fails if the given objects are unequal.

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

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

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

If `strip_spaces` is given a true value (see *Boolean arguments*) and both arguments are strings, the comparison is done without leading and trailing spaces. If `strip_spaces` is given a string value `LEADING` or `TRAILING` (case-insensitive), the comparison is done without leading or trailing spaces, respectively.

If `collapse_spaces` is given a true value (see *Boolean arguments*) and both arguments are strings, the comparison is done with all white spaces replaced by a single space character.

`strip_spaces` is new in Robot Framework 4.0 and `collapse_spaces` is new in Robot Framework 4.1.

**should\_be\_equal\_as\_integers** (*first, second, msg=None, values=True, base=None*)

Fails if objects are unequal after converting them to integers.

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

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

**should\_be\_equal\_as\_numbers** (*first, second, msg=None, values=True, precision=6*)

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

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

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

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

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

**should\_be\_equal\_as\_strings** (*first, second, msg=None, values=True, ignore\_case=False, strip\_spaces=False, formatter='str', collapse\_spaces=False*)

Fails if objects are unequal after converting them to strings.

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

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

If `strip_spaces` is given a true value (see *Boolean arguments*) and both arguments are strings, the comparison is done without leading and trailing spaces. If `strip_spaces` is given a string value `LEADING` or `TRAILING` (case-insensitive), the comparison is done without leading or trailing spaces, respectively.

If `collapse_spaces` is given a true value (see *Boolean arguments*) and both arguments are strings, the comparison is done with all white spaces replaced by a single space character.

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

`strip_spaces` is new in Robot Framework 4.0 and `collapse_spaces` is new in Robot Framework 4.1.

**should\_be\_true** (*condition, msg=None*)

Fails if the given condition is not true.

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

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

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

**should\_contain** (*container*, *item*, *msg=None*, *values=True*, *ignore\_case=False*, *strip\_spaces=False*, *collapse\_spaces=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.

If `strip_spaces` is given a true value (see *Boolean arguments*) and both arguments are strings, the comparison is done without leading and trailing spaces. If `strip_spaces` is given a string value `LEADING` or `TRAILING` (case-insensitive), the comparison is done without leading or trailing spaces, respectively.

If `collapse_spaces` is given a true value (see *Boolean arguments*) and both arguments are strings, the comparison is done with all white spaces replaced by a single space character.

`strip_spaces` is new in Robot Framework 4.0 and `collapse_spaces` is new in Robot Framework 4.1.

**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`, `ignore_case` and `strip_spaces`, and `collapse_spaces` 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`.

**should\_contain\_x\_times** (*container*, *item*, *count*, *msg=None*, *ignore\_case=False*, *strip\_spaces=False*, *collapse\_spaces=False*)

Fails if `container` does not contain `item` `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 `container` is a list-like object, string items in it are compared case-insensitively.

If `strip_spaces` is given a true value (see *Boolean arguments*) and both arguments are strings, the comparison is done without leading and trailing spaces. If `strip_spaces` is given a string value `LEADING` or `TRAILING` (case-insensitive), the comparison is done without leading or trailing spaces, respectively.

If `collapse_spaces` is given a true value (see *Boolean arguments*) and both arguments are strings, the comparison is done with all white spaces replaced by a single space character.



`strip_spaces` is new in Robot Framework 4.0 and `collapse_spaces` is new in Robot Framework 4.1.

**should\_end\_with** (*str1*, *str2*, *msg=None*, *values=True*, *ignore\_case=False*, *strip\_spaces=False*, *collapse\_spaces=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`, `strip_spaces`, and `collapse_spaces` options.

**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.

If `ignore_case` is given a true value (see *Boolean arguments*) and compared items are strings, it indicates that comparison should be case-insensitive.

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

**should\_match\_regexp** (*string*, *pattern*, *msg=None*, *values=True*)

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

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 `(?im)pattern`. The most useful flags are `i` (case-insensitive), `m` (multiline mode), `s` (dotall mode) and `x` (verbose).

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

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

**should\_not\_be\_empty** (*item*, *msg=None*)

Verifies that the given item is not empty.

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

**should\_not\_be\_equal** (*first*, *second*, *msg=None*, *values=True*, *ignore\_case=False*, *strip\_spaces=False*, *collapse\_spaces=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.

If `strip_spaces` is given a true value (see *Boolean arguments*) and both arguments are strings, the comparison is done without leading and trailing spaces. If `strip_spaces` is given a string value `LEADING` or `TRAILING` (case-insensitive), the comparison is done without leading or trailing spaces, respectively.

If `collapse_spaces` is given a true value (see *Boolean arguments*) and both arguments are strings, the comparison is done with all white spaces replaced by a single space character.

`strip_spaces` is new in Robot Framework 4.0 and `collapse_spaces` is new in Robot Framework 4.1.

**`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, strip\_spaces=False, collapse\_spaces=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.

If `strip_spaces` is given a true value (see *Boolean arguments*) and both arguments are strings, the comparison is done without leading and trailing spaces. If `strip_spaces` is given a string value `LEADING` or `TRAILING` (case-insensitive), the comparison is done without leading or trailing spaces, respectively.

If `collapse_spaces` is given a true value (see *Boolean arguments*) and both arguments are strings, the comparison is done with all white spaces replaced by a single space character.

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

`strip_spaces` is new in Robot Framework 4.0 and `collapse_spaces` is new in Robot Framework 4.1.

**`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, strip\_spaces=False, collapse\_spaces=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*.

If `strip_spaces` is given a true value (see *Boolean arguments*) and both arguments are strings, the comparison is done without leading and trailing spaces. If `strip_spaces` is given a string value `LEADING` or `TRAILING` (case-insensitive), the comparison is done without leading or trailing spaces, respectively.

If `collapse_spaces` is given a true value (see *Boolean arguments*) and both arguments are strings, the comparison is done with all white spaces replaced by a single space character.

`strip_spaces` is new in Robot Framework 4.0 and `collapse_spaces` is new in Robot Framework 4.1.

**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`, `ignore_case` and `strip_spaces`, and `collapse_spaces` 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*.

**should\_not\_end\_with** (*str1*, *str2*, *msg=None*, *values=True*, *ignore\_case=False*,  
*strip\_spaces=False*, *collapse\_spaces=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`, `strip_spaces`, and `collapse_spaces` options.

**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.

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

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

**should\_not\_match\_regex** (*string*, *pattern*, *msg=None*, *values=True*)

Fails if *string* matches *pattern* as a regular expression.

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

**should\_not\_start\_with** (*str1*, *str2*, *msg=None*, *values=True*, *ignore\_case=False*,  
*strip\_spaces=False*, *collapse\_spaces=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`, `strip_spaces`, and `collapse_spaces` options.

**should\_start\_with** (*str1*, *str2*, *msg=None*, *values=True*, *ignore\_case=False*, *strip\_spaces=False*,  
*collapse\_spaces=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`, `strip_spaces`, and `collapse_spaces` options.

**skip** (*msg='Skipped with Skip keyword.'*)

Skips the rest of the current test.

Skips the remaining keywords in the current test and sets the given message to the test. If the test has teardown, it will be executed.

**skip\_if** (*condition*, *msg=None*)

Skips the rest of the current test if the `condition` is `True`.

Skips the remaining keywords in the current test and sets the given message to the test. If `msg` is not given, the `condition` will be used as the message. If the test has teardown, it will be executed.

If the `condition` evaluates to `False`, does nothing.

**sleep** (*time\_*, *reason=None*)

Pauses the test executed for the given time.

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

**variable\_should\_exist** (*name*, *msg=None*)

Fails unless the given variable exists within the current scope.

The name of the variable can be given either as a normal variable name like `${name}` or in escaped format like `$name` or `\${name}`. For the reasons explained in the *Using variables with keywords creating or accessing variables* section, using the escaped format is recommended.

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 like `${name}` or in escaped format like `$name` or `\${name}`. For the reasons explained in the *Using variables with keywords creating or accessing variables* section, using the escaped format is recommended.

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 between execution attempts.

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.

By default `retry_interval` is the time to wait `_after_` a keyword has failed. For example, if the first run takes 2 seconds and the retry interval is 3 seconds, the second run starts 5 seconds after the first run started. If `retry_interval` start with prefix `strict:`, the execution time of the previous keyword is subtracted from the retry time. With the earlier example the second run would thus start 3 seconds after the first run started. A warning is logged if keyword execution time is longer than a strict interval.

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

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

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

Support for “strict” retry interval is new in Robot Framework 4.1.

**exception** `robot.libraries.BuiltIn.RobotNotRunningError`

Bases: `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**

**with\_traceback()**

`Exception.with_traceback(tb)` – set `self.__traceback__` to `tb` and return `self`.

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

Tell Robot Framework that this keyword runs other keywords internally.

*NOTE:* This API will change in the future. 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 using *Run Keyword* or its variants like *Run Keyword If* need some special handling by the framework. This includes not processing arguments (e.g. variables in them) twice, special handling of timeouts, and so on.

2) How to use this method

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

*keyword* is the name of the keyword. With Python 2 it is possible to pass also the function or method implementing the keyword.

*args\_to\_process* defines how many of the arguments to the registered keyword must be processed normally.

3) Examples

```
from robot.libraries.BuiltIn import BuiltIn, register_run_keyword
```

```
def my_run_keyword(name, *args): # do something return BuiltIn().run_keyword(name, *args)
```

```
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)
```

```
register_run_keyword('MyLibrary', 'my_run_keyword_if', 2)
```

## robot.libraries.Collections module

**class** `robot.libraries.Collections.NotSet`

Bases: `object`

**class** robot.libraries.Collections.Collections

Bases: robot.libraries.Collections.\_List, robot.libraries.Collections.\_Dictionary

A 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*).

== Table of contents ==

%TOC%

= 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 extent 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#truthrules> as in Python].

True examples:

False examples:

Considering `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 = '5.0.1'**

**should\_contain\_match** (*list*, *pattern*, *msg=None*, *case\_insensitive=False*, *whitespace\_insensitive=False*)

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

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

If you prepend `regexp=` to your pattern, your pattern will be used according to the Python [<http://docs.python.org/library/re.html#re> module] regular expression syntax. Important note: Backslashes are

an escape character, and must be escaped with another backslash (e.g. `regexp=\\d{6}` to search for `\\d{6}`). See *BuiltIn.Should Match Regexp* for more details.

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

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

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

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

See also *Should Not Contain Match*.

**should\_not\_contain\_match** (*list*, *pattern*, *msg=None*, *case\_insensitive=False*, *whitespace\_insensitive=False*)

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

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

**get\_matches** (*list*, *pattern*, *case\_insensitive=False*, *whitespace\_insensitive=False*)

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

For more information on *pattern*, *case\_insensitive*, and *whitespace\_insensitive*, see *Should Contain Match*.

**get\_match\_count** (*list*, *pattern*, *case\_insensitive=False*, *whitespace\_insensitive=False*)

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

For more information on *pattern*, *case\_insensitive*, and *whitespace\_insensitive*, see *Should Contain Match*.

**append\_to\_list** (*list*, *\*values*)

Adds *values* to the end of *list*.

**combine\_lists** (*\*lists*)

Combines the given *lists* together and returns the result.

The given lists are not altered by this keyword.

**convert\_to\_dictionary** (*item*)

Converts the given *item* to a Python dict type.

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

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

**convert\_to\_list** (*item*)

Converts the given *item* to a Python list type.

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

**copy\_dictionary** (*dictionary*, *deepcopy=False*)

Returns a copy of the given dictionary.

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

The given dictionary is never altered by this keyword.

**copy\_list** (*list\_*, *deepcopy=False*)

Returns a copy of the given list.

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

The given list is never altered by this keyword.

**count\_values\_in\_list** (*list\_*, *value*, *start=0*, *end=None*)

Returns the number of occurrences of the given *value* in *list*.

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

**dictionaries\_should\_be\_equal** (*dict1*, *dict2*, *msg=None*, *values=True*)

Fails if the given dictionaries are not equal.

First the equality of dictionaries' keys is checked and after that all the key value pairs. If there are differences between the values, those are listed in the error message. The types of the dictionaries do not need to be same.

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

**dictionary\_should\_contain\_item** (*dictionary*, *key*, *value*, *msg=None*)

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

Value is converted to unicode for comparison.

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

**dictionary\_should\_contain\_key** (*dictionary*, *key*, *msg=None*)

Fails if *key* is not found from dictionary.

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

**dictionary\_should\_contain\_sub\_dictionary** (*dict1*, *dict2*, *msg=None*, *values=True*)

Fails unless all items in *dict2* are found from *dict1*.

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

**dictionary\_should\_contain\_value** (*dictionary*, *value*, *msg=None*)

Fails if *value* is not found from dictionary.

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

**dictionary\_should\_not\_contain\_key** (*dictionary*, *key*, *msg=None*)

Fails if *key* is found from dictionary.

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

**dictionary\_should\_not\_contain\_value** (*dictionary*, *value*, *msg=None*)

Fails if *value* is found from dictionary.

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

**get\_dictionary\_items** (*dictionary*, *sort\_keys=True*)

Returns items of the given dictionary as a list.

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



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

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

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

**get\_dictionary\_keys** (*dictionary*, *sort\_keys=True*)

Returns keys of the given `dictionary` as a list.

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

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

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

**get\_dictionary\_values** (*dictionary*, *sort\_keys=True*)

Returns values of the given `dictionary` as a list.

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

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

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

**get\_from\_dictionary** (*dictionary*, *key*)

Returns a value from the given `dictionary` based on the given `key`.

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

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

**get\_from\_list** (*list\_*, *index*)

Returns the value specified with an `index` from `list`.

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

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

**get\_index\_from\_list** (*list\_*, *value*, *start=0*, *end=None*)

Returns the index of the first occurrence of the `value` on the list.

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

**get\_slice\_from\_list** (*list\_*, *start=0*, *end=None*)

Returns a slice of the given `list` between `start` and `end` indexes.

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

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

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

**insert\_into\_list** (*list\_*, *index*, *value*)

Inserts *value* into *list* to the position specified with *index*.

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

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

**keep\_in\_dictionary** (*dictionary*, *\*keys*)

Keeps the given keys in the *dictionary* and removes all other.

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

**list\_should\_contain\_sub\_list** (*list1*, *list2*, *msg=None*, *values=True*)

Fails if not all of the elements in *list2* are found in *list1*.

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

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

**list\_should\_contain\_value** (*list\_*, *value*, *msg=None*)

Fails if the *value* is not found from *list*.

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

**list\_should\_not\_contain\_duplicates** (*list\_*, *msg=None*)

Fails if any element in the *list* is found from it more than once.

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

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

**list\_should\_not\_contain\_value** (*list\_*, *value*, *msg=None*)

Fails if the *value* is found from *list*.

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

**lists\_should\_be\_equal** (*list1*, *list2*, *msg=None*, *values=True*, *names=None*, *ignore\_order=False*)

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*.

The 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.

The optional `ignore_order` argument can be used to ignore the order of the elements in the lists. Using it requires items to be sortable. This is new in Robot Framework 3.2.

**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.

**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 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 ==

%TOC%

= Terminology =

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

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

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

= Date formats =

Dates can be 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](http://en.wikipedia.org/wiki/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.

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

Notice that epoch times are independent on time zones and thus same around the world at a certain time. For example, epoch times returned by *Get Current Date* are not affected by the `time_zone` argument. What local time a certain epoch time matches then depends on the time zone.

Following examples demonstrate using epoch times. They are tested in Finland, and due to the reasons explained above they would fail on other time zones.

### == Earliest supported date ==

The earliest date that is supported depends on the date format and to some extent 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.
- On other platforms epoch time supports 1900 and above or even earlier.

### = 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>] objects are also supported both in input and in output. In input they are recognized automatically, and in output it is possible to receive them by giving `timedelta` value to `result_format` argument.

= Millisecond handling =

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

All keywords returning dates or times have an option to leave milliseconds out by giving a true value to `exclude_millis` argument. If the argument is given as a string, it is considered true unless it is empty or case-insensitively equal to `false`, `none` or `no`. Other argument types are tested using same [<http://docs.python.org/library/stdtypes.html#truthrules> as in Python].

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

= Programmatic usage =

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

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

```
robot.libraries.DateTime.get_current_date (time_zone='local',          increment=0,
                                         result_format='timestamp',    ex-
                                         clude_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. Has no effect if date is returned as an *epoch time*.

- **increment**: Optional time increment to add to the returned date in one of the supported *time formats*. Can be negative.
- **result\_format**: Format of the returned date (see *date formats*).
- **exclude\_millis**: When set to any true value, rounds and drops milliseconds as explained in *millisecond handling*.

```
robot.libraries.DateTime.convert_date (date,          result_format='timestamp',    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.

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

Converts between supported *time formats*.

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

milliseconds as explained in *millisecond handling*.

```
robot.libraries.DateTime.subtract_date_from_date (date1,          date2,          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 *millisecond handling*.
- **date1\_format**: Possible *custom timestamp* format of `date1`.
- **date2\_format**: Possible *custom timestamp* format of `date2`.

Examples:

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

Adds time to date and returns the resulting date.

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

*date formats*.

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

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

Subtracts time from date and returns the resulting date.

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

*date formats*.

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

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

Adds time to another time and returns the resulting time.

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

milliseconds as explained in *millisecond handling*.

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

Subtracts time from another time and returns the resulting time.

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

the supported *time formats*.

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



## robot.libraries.Dialogs module

A library providing dialogs for interacting with users.

Dialogs is Robot Framework's standard library that provides means for pausing the test or task execution and getting input from users.

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.

`robot.libraries.Dialogs.pause_execution` (*message='Execution paused. Press OK to continue.'*)

Pauses execution until user clicks Ok button.

`message` is the message shown in the dialog.

`robot.libraries.Dialogs.execute_manual_step` (*message, default\_error=""*)

Pauses execution until user sets the keyword status.

User can press either PASS or FAIL button. In the latter case execution fails and an additional dialog is opened for defining the error message.

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

`robot.libraries.Dialogs.get_value_from_user` (*message, default\_value="", hidden=False*)

Pauses 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].

`robot.libraries.Dialogs.get_selection_from_user` (*message, \*values*)

Pauses execution and asks user to select a value.

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

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

`robot.libraries.Dialogs.get_selections_from_user` (*message, \*values*)

Pauses 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.

## robot.libraries.Easter module

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

## robot.libraries.OperatingSystem module

**class** robot.libraries.OperatingSystem.OperatingSystem

Bases: object

A library providing keywords for operating system 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 ==

%TOC%

= Path separators =

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

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

= Pattern matching =

Many keywords accepts arguments as either `_glob_` or `_regular expression_` patterns.

== Glob patterns ==

Some keywords, for example *List Directory*, support so called [[http://en.wikipedia.org/wiki/Glob\\_\(programming\)](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.

== Regular expressions ==

Some keywords, for example *Grep File*, support [[http://en.wikipedia.org/wiki/Regular\\_expression](http://en.wikipedia.org/wiki/Regular_expression)]regular expressions] that are more powerful but also more complicated than 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 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 from the BuiltIn library.

= Tilde expansion =

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

= 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:

= Example =

```
ROBOT_LIBRARY_SCOPE = 'GLOBAL'
```

```
ROBOT_LIBRARY_VERSION = '5.0.1'
```

**run** (*command*)

Runs the given command in the system and returns the output.

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

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

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

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

**run\_and\_return\_rc** (*command*)

Runs the given command in the system and returns the return code.

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

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

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

**run\_and\_return\_rc\_and\_output** (*command*)

Runs the given command in the system and returns the RC and output.

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

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

**get\_file** (*path*, *encoding*='UTF-8', *encoding\_errors*='strict')

Returns the contents of a specified file.

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

*encoding* defines the encoding of the file. The default value is UTF-8, which means that UTF-8 and ASCII encoded files are read correctly. In addition to the encodings supported by the underlying Python implementation, the following special encoding values can be used:

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

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

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

**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', *regexp*=False)

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 concatenated 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 i.e. it does not need to match the pattern fully. There are two supported pattern types:

- By default the pattern is considered a **\_glob\_** pattern where, for example, \* and ? can be used as wildcards.
- If the *regexp* argument is given a true value, the pattern is considered to be a **\_regular expression\_**. These patterns are more powerful but also more complicated than glob patterns. They often use the backslash character and it needs to be escaped in Robot Framework date like \.

For more information about glob and regular expression syntax, see the *Pattern matching* section. With this keyword matching is always case-sensitive.

Special encoding values **SYSTEM** and **CONSOLE** that *Get File* supports are supported by this keyword only with Robot Framework 4.0 and newer.

Support for regular expressions is new in Robot Framework 5.0.

**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. See the *Glob patterns* section for details about the supported syntax.

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. See the *Glob patterns* section for details about the supported syntax.

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. See the *Glob patterns* section for details about the supported syntax.

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. See the *Glob patterns* section for details about the supported syntax.

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. See the *Glob patterns* section for details about the supported syntax.

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. See the *Glob patterns* section for details about the supported syntax.

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. See the *Glob patterns* section for details about the supported syntax. 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. See the *Glob patterns* section for details about the supported syntax. 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 file 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.

**`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.

**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. See the *Glob patterns* section for details about the supported syntax. 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 *Glob patterns*) 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.

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.

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 *Glob patterns*). 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 environment variable is found, returns possible default value. If no default value is given, the keyword fails.

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.
- If `case_normalize` is given a true value (see *Boolean arguments*) on Windows, converts the path to all lowercase.
- `${path1}` = `'abc'`
- `${path2}` = `'def'`
- `${path3}` = `'abc/def/ghi'`
- `${path4}` = `'/home/robot/stuff'`

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

**split\_path** (*path*)

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

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

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

**split\_extension** (*path*)

Splits the extension from the given path.

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

- `${path} = 'file' & ${ext} = 'extension'`
- `${p2} = 'path/file' & ${e2} = 'ext'`
- `${p3} = 'path/file' & ${e3} = ''`
- `${p4} = 'p2/file' & ${e4} = 'ext'`
- `${p5} = 'path/.file' & ${e5} = 'ext'`
- `${p6} = 'path/.file' & ${e6} = ''`

**get\_modified\_time** (*path*, *format*='timestamp')

Returns the last modification time of a file or directory.

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

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

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

**set\_modified\_time** (*path*, *mtime*)

Sets the file modification and access times.

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

- 1) If `mtime` is a number, or a string that can be converted to a number, it is interpreted as seconds since the UNIX epoch (1970-01-01 00:00:00 UTC). This documentation was originally written about 1177654467 seconds after the epoch.
- 2) If `mtime` is a timestamp, that time will be used. Valid timestamp formats are `YYYY-MM-DD hh:mm:ss` and `YYYYMMDD hhmmss`.

- 3) If `mtime` is equal to `NOW`, the current local time is used.
- 4) If `mtime` is equal to `UTC`, the current time in [\[http://en.wikipedia.org/wiki/Coordinated\\_Universal\\_Time|UTC\]](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 is considered to be a *\_glob pattern\_* and the full syntax is explained in the *Glob patterns* section. With this keyword matching is always 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 library for running processes.

This library utilizes Python's [\[http://docs.python.org/library/subprocess.html#subprocess\]](http://docs.python.org/library/subprocess.html#subprocess) module and its [\[http://docs.python.org/library/subprocess.html#popen-constructors\]](http://docs.python.org/library/subprocess.html#popen-constructors) `Popen` class.

The library has following main usages:

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

== Table of contents ==

%TOC%

= 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 Robot Framework, 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.

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.

#### == Standard input stream ==

The `stdin` argument makes it possible to pass information to the standard input stream of the started process. How its value is interpreted is explained in the table below.

Values `PIPE` and `NONE` are internally mapped directly to `subprocess.PIPE` and `None`, respectively, when calling [<https://docs.python.org/3/library/subprocess.html#subprocess.Popen>]. The default behavior may change from `PIPE` to `NONE` in future releases. If you depend on the `PIPE` behavior, it is a good idea to use it explicitly.

The support to configure `stdin` is new in Robot Framework 4.1.2.

#### == 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 than configured, values got through the *result object* will be invalid.

#### == Alias ==

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

= Active process =

The library keeps record which of the started processes is currently active. By default it is the latest process started with *Start Process*, but *Switch Process* can be used to activate a different process. 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 an `alias` explicitly given to *Start Process* or the process object returned by it.

= 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#truthrules> as in Python].

True examples:

False examples:

Considering `OFF` and `0` false is new in Robot Framework 3.1.

= Example =

```
ROBOT_LIBRARY_SCOPE = 'GLOBAL'
```

```
ROBOT_LIBRARY_VERSION = '5.0.1'
```

```
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 the created [<https://docs.python.org/3/library/subprocess.html#popen-constructor> | `subprocess.Popen`] object which can be used later to active this process. `Popen` attributes like `pid` can also be accessed directly.

Processes are started so that they create a new process group. This allows terminating and sending signals to possible child processes.

Start process and wait for it to end later using alias:

Use returned `Popen` object:

Use started process in a pipeline with another process:

Returning a `subprocess.Popen` object is new in Robot Framework 5.0. Earlier versions returned a generic handle and getting the process object required using *Get Process Object* separately.

**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: - 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*).

**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*.

Starting from Robot Framework 5.0, it is also possible to directly access the *pid* attribute of the `subprocess.Popen` object returned by *Start Process* like `${process.pid}`.

**get\_process\_object** (*handle=None*)

Return the underlying `subprocess.Popen` object.

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



Starting from Robot Framework 5.0, *Start Process* returns the created `subprocess.Popen` object, not a generic handle, making this keyword mostly redundant.

**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 finished before this keyword can be used. In practice this means that processes started with *Start Process* must be finished either with *Wait For Process* or *Terminate Process* before using this keyword.

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

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

**switch\_process** (*handle*)

Makes the specified process the current *active process*.

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

**split\_command\_line** (*args, escaping=False*)

Splits command line string into a list of arguments.

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

**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.

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

Bases: object

**stdout**

**stderr**

**close\_streams** ()

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

Bases: object

**get\_command** (*command, arguments*)

**popen\_config**

**result\_config**

**robot.libraries.Remote module**

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

Bases: object

Connects to a remote server at *uri*.

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

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

```
ROBOT_LIBRARY_SCOPE = 'TEST SUITE'
```

```
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.ArgumentCoercer
```

Bases: object

```
binary = re.compile('[\x00-\x08\x0b\x0c\x0e-\x1f]')
```

```
non_ascii = re.compile('[\x80-\xff]')
```

```
coerce (argument)
```

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

Bases: object

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

Bases: object

```
get_library_information ()
```

```
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: xmlrpc.client.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=False)
    send_content (connection, request_body)
    send_headers (connection, headers)
    send_request (host, handler, request_body, debug)
    single_request (host, handler, request_body, verbose=False)
    user_agent = 'Python-xmlrpc/3.7'

class robot.libraries.Remote.TimeoutHTTPSTransport (use_datetime=0, timeout=None)
    Bases: robot.libraries.Remote.TimeoutHTTPTransport
    accept_gzip_encoding = True
    close ()
    encode_threshold = None
    get_host_info (host)
    getparser ()
    make_connection (host)
    parse_response (response)
    request (host, handler, request_body, verbose=False)
    send_content (connection, request_body)
    send_headers (connection, headers)
    send_request (host, handler, request_body, debug)
    single_request (host, handler, request_body, verbose=False)
    user_agent = 'Python-xmlrpc/3.7'

```

### robot.libraries.Reserved module

```

class robot.libraries.Reserved.Reserved
    Bases: object
    ROBOT_LIBRARY_SCOPE = 'GLOBAL'

```

### robot.libraries.Screenshot module

```

class robot.libraries.Screenshot.Screenshot (screenshot_directory=None,          screen-
                                             shot_module=None)
    Bases: object

```

Library for taking screenshots on the machine where tests are executed.

Taking the actual screenshot requires a suitable tool or module that may need to be installed separately. Taking screenshots also requires tests to be run with a physical or virtual display.

== Table of contents ==

%TOC%

= Supported screenshot taking tools and modules =

How screenshots are taken 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> :: Generic Python GUI toolkit.
- 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.

= 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 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.

**ROBOT\_LIBRARY\_SCOPE = 'TEST SUITE'**

**ROBOT\_LIBRARY\_VERSION = '5.0.1'**

**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 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** = '5.0.1'

**convert\_to\_lower\_case** (*string*)

Converts string to lower case.

Uses Python's standard [<https://docs.python.org/library/stdtypes.html#str.lower>] method.

**convert\_to\_upper\_case** (*string*)

Converts string to upper case.

Uses Python's standard [<https://docs.python.org/library/stdtypes.html#str.upper>] method.

**convert\_to\_title\_case** (*string*, *exclude=None*)

Converts string to title case.

Uses the following algorithm:

- Split the string to words from whitespace characters (spaces, newlines, etc.).
- Exclude words that are not all lower case. This preserves, for example, “OK” and “iPhone”.
- Exclude also words listed in the optional `exclude` argument.
- Title case the first alphabetical character of each word that has not been excluded.
- Join all words together so that original whitespace is preserved.

Explicitly excluded words can be given as a list or as a string with words separated by a comma and an optional space. Excluded words are actually considered to be regular expression patterns, so it is possible to use something like “example[.!]?” to match the word “example” on it own and also if followed by “.”, “!” or “?”. See *BuiltIn.Should Match Regexp* for more information about Python regular expression syntax in general and how to use it in Robot Framework data in particular.

The reason this keyword does not use Python’s standard [<https://docs.python.org/library/stdtypes.html#str.title>] method is that it can yield undesired results, for example, if strings contain upper case letters or special characters like apostrophes. It would, for example, convert “it’s an OK iPhone” to “It’S An Ok Iphone”.

New in Robot Framework 3.2.

**encode\_string\_to\_bytes** (*string*, *encoding*, *errors='strict'*)

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

*errors* argument controls what to do if encoding some characters fails. All values accepted by `encode` method in Python are valid, but in practice the following values are most useful:

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

Use *Convert To Bytes* in *BuiltIn* if you want to create bytes based on character or integer sequences. Use *Decode Bytes To String* if you need to convert byte strings to Unicode strings and *Convert To String* in *BuiltIn* if you need to convert arbitrary objects to Unicode.

**decode\_bytes\_to\_string** (*bytes*, *encoding*, *errors='strict'*)

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

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

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

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

**format\_string** (*template*, *\*positional*, *\*\*named*)

Formats a *template* using the given *positional* and *named* arguments.

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

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

New in Robot Framework 3.1.

**get\_line\_count** (*string*)

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

**split\_to\_lines** (*string*, *start=0*, *end=None*)

Splits the given string to lines.

It is possible to get only a selection of lines from *start* to *end* so that *start* index is inclusive and *end* is exclusive. Line numbering starts from 0, and it is possible to use negative indices to refer to lines from the end.

Lines are returned without the newlines. The number of returned lines is automatically logged.

Use *Get Line* if you only need to get a single line.

**get\_line** (*string*, *line\_number*)

Returns the specified line from the given *string*.

Line numbering starts from 0 and it is possible to use negative indices to refer to lines from the end. The line is returned without the newline character.

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

**get\_lines\_containing\_string** (*string*, *pattern*, *case\_insensitive=False*)

Returns lines of the given *string* that contain the *pattern*.

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

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

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.

Lines are returned as one string catenated back together with newlines. Possible trailing newline is never returned. The number of matching lines is automatically logged.

See *Get Lines Matching Regexp* if you need more complex patterns and *Get Lines Containing String* if searching literal strings is enough.

**get\_lines\_matching\_regexp** (*string*, *pattern*, *partial\_match=False*)

Returns lines of the given *string* that match the regexp *pattern*.

See *BuiltIn.Should Match Regexp* for more information about Python regular expression syntax in general and how to use it in Robot Framework 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.

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).

**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 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.

**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 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`.

`length` can be given as a number, a string representation of a number, or as a range of numbers, such as 5-10. When a range of values is given the range will be selected by random within the range.

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:

Giving `length` as a range of values is new in Robot Framework 5.0.

**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.

**should\_be\_string** (*item*, *msg=None*)

Fails if the given *item* is not a 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.

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.

On Python 3 this keyword behaves exactly the same way *Should Be String*. That keyword should be used instead and this keyword will be deprecated.

**should\_be\_byte\_string** (*item*, *msg=None*)

Fails if the given *item* is not a byte string.

Use *Should Be String* if you want to verify the *item* is a string.

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

**should\_be\_lower\_case** (*string*, *msg=None*)

Fails if the given *string* is not in lower case.

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 Upper Case* and *Should Be Title Case*.

**should\_be\_upper\_case** (*string*, *msg=None*)

Fails if the given *string* is not in upper case.

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 Title Case* and *Should Be Lower Case*.

**should\_be\_title\_case** (*string*, *msg=None*, *exclude=None*)

Fails if given *string* is not title.

*string* is a title cased string if there is at least one upper case letter in each word.

For example, 'This Is Title' and 'OK, Give Me My iPhone' would pass. 'all words lower' and 'Word In lower' would fail.

This logic changed in Robot Framework 4.0 to be compatible with *Convert to Title Case*. See *Convert to Title Case* for title case algorithm and reasoning.

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

Words can be explicitly excluded with the optional *exclude* argument.

Explicitly excluded words can be given as a list or as a string with words separated by a comma and an optional space. Excluded words are actually considered to be regular expression patterns, so it is possible to use something like "example[.!]?)" to match the word "example" on it own and also if followed by ":", "!" or "?". See *BuiltIn.Should Match Regexp* for more information about Python regular expression syntax in general and how to use it in Robot Framework data in particular.

See also *Should Be Upper Case* and *Should Be Lower Case*.

## robot.libraries.Telnet module

```
class robot.libraries.Telnet.Telnet (timeout='3 seconds', newline='CRLF', prompt=None,
                                     prompt_is_regex=False, encoding='UTF-8', en-
                                     coding_errors='ignore', default_log_level='INFO',
                                     window_size=None, environ_user=None, termi-
                                     nal_emulation=False, terminal_type=None, tel-
                                     netlib_log_level='TRACE', connection_timeout=None)
```

Bases: object

A 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 ==

%TOC%

= 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.

#### == 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 data is passed to the library as Unicode and you need to use variables to use bytes.

It is also possible to configure the error handler to use if encoding or decoding characters fails. Accepted values are the same that encode/decode functions in Python strings accept. In practice the following values are the most useful:

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

#### == Default log level ==

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

#### == Terminal type ==

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

#### == Window size ==

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

#### == USER environment variable ==

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

#### = Terminal emulation =

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

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

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

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

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

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

#### = Logging =

All keywords that read something log the output. These keywords take the log level to use as an optional argument, and if no log level is specified they use the [`#Configuration|configured`] default value.

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

The [<http://docs.python.org/library/telnetlib.html#telnetlib> module] used by this library has a custom logging system for logging content it sends and receives. By default these messages are written using `TRACE` level, but the level is configurable with the `telnetlib_log_level` option either in the library initialization, to the *Open Connection* or by using the *Set Telnetlib Log Level* keyword to the active connection. Special level `NONE` 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#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.

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 = 'SUITE'
```

```
ROBOT_LIBRARY_VERSION = '5.0.1'
```

```
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, de-
                fault_log_level=None, window_size=None, environ_user=None, termi-
                nal_emulation=None, terminal_type=None, telnetlib_log_level=None, con-
                nection_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, time-
                                              out=3.0, newline='CRLF',
                                              prompt=None, prompt_is_regexp=False,
                                              encoding='UTF-8', encoding_
                                              errors='ignore', de-
                                              fault_log_level='INFO', win-
                                              dow_size=None, environ_user=None,
                                              terminal_emulation=False,
                                              terminal_type=None, tel-
                                              netlib_log_level='TRACE', connec-
                                              tion_timeout=None)
```

Bases: `telnetlib.Telnet`

**NEW\_ENVIRON\_IS** = `b'\x00'`

**NEW\_ENVIRON\_VAR** = `b'\x00'`

**NEW\_ENVIRON\_VALUE** = `b'\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 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 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*)

Print a debug message, when the debug level is > 0.

If extra arguments are present, they are substituted in the message using the standard string formatting operator.

**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.Pattern` 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 `re.Match` object returned; and the text read up till and including the match.

If EOF is read and no text was read, raise `EOFError`. Otherwise, when nothing matches, return `(-1, None, text)` where `text` is the text received so far (may be the empty string if a timeout happened).

If a regular expression ends with a greedy match (e.g. `.*`) or if more than one expression can match the same input, the results are undeterministic, and may depend on the I/O timing.

**fileno** ()

Return the `fileno()` of the socket object used internally.

**fill\_rawq** ()

Fill raw queue from exactly one `recv()` system call.

Block if no data is immediately available. Set `self.eof` when connection is closed.

**get\_socket** ()

Return the socket object used internally.

**interact** ()

Interaction function, emulates a very dumb telnet client.

**listener** ()

Helper for `mt_interact()` – this executes in the other thread.

**mt\_interact** ()

Multithreaded version of `interact()`.

**open** (*host*, *port=0*, *timeout=<object object>*)

Connect to a host.

The optional second argument is the port number, which defaults to the standard telnet port (23).

Don't try to reopen an already connected instance.

**process\_rawq** ()

Transfer from raw queue to cooked queue.

Set `self.eof` when connection is closed. Don't block unless in the midst of an IAC sequence.

**rawq\_getchar()**  
Get next char from raw queue.  
Block if no data is immediately available. Raise EOFError when connection is closed.

**read\_all()**  
Read all data until EOF; block until connection closed.

**read\_eager()**  
Read readily available data.  
Raise EOFError if connection closed and no cooked data available. Return b'' if no cooked data available otherwise. Don't block unless in the midst of an IAC sequence.

**read\_lazy()**  
Process and return data that's already in the queues (lazy).  
Raise EOFError if connection closed and no data available. Return b'' 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 b'' 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 b'' 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 b'' 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 b'' if no cooked data available otherwise. Don't block.

**set\_debuglevel(*debuglevel*)**  
Set the debug level.  
The higher it is, the more debug output you get (on sys.stdout).

**set\_option\_negotiation\_callback(*callback*)**  
Provide a callback function called after each receipt of a telnet option.

**sock\_avail()**  
Test whether data is available on the socket.

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

**current\_output**

**feed(*text*)**

**read()**

**read\_until(*expected*)**

**read\_until\_regexp(*regexp\_list*)**

**exception** `robot.libraries.Telnet.NoMatchError` (*expected, timeout, output=None*)

Bases: `AssertionError`

**ROBOT\_SUPPRESS\_NAME** = `True`

**args**

**with\_traceback()**

Exception.with\_traceback(tb) – set self.\_\_traceback\_\_ to tb and return self.

## robot.libraries.XML module

**class** `robot.libraries.XML.XML` (*use\_lxml=False*)

Bases: `object`

Robot Framework library for verifying and modifying XML documents.

As the name implies, `_XML_` is a library for verifying contents of XML files. In practice, it is a pretty thin wrapper on top of Python's [<http://docs.python.org/library/xml.etree.elementtree.html>] `ElementTree` XML API.

The library has the following main usages:

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

== Table of contents ==

%TOC%

= 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 be 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 in the 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/lxml>]

module instead when *importing* the library. The resulting element structure has same API regardless which module is used for parsing.

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

= Example =

The following simple example demonstrates parsing XML and verifying its contents both using keywords in this library and in `_BuiltIn_` and `_Collections_` libraries. How to use xpath expressions to find elements and what attributes the returned elements contain are discussed, with more examples, in *Finding elements with xpath* and *Element attributes* sections.

In this example, as well as in many other examples in this documentation, `{XML}` refers to the following example XML document. In practice `{XML}` could either be a path to an XML file or it could contain the XML itself.

Notice that in the example three last lines are equivalent. Which one to use in practice depends on which other elements you need to get or verify. If you only need to do one verification, using the last line alone would suffice. If more verifications are needed, parsing the XML with *Parse XML* only once would be more efficient.

= Finding elements with xpath =

`ElementTree`, and thus also this library, supports finding elements using xpath expressions. `ElementTree` does not, however, support the full xpath standard. The supported xpath syntax is explained below and [<https://docs.python.org/library/xml.etree.elementtree.html#xpath-support>] `ElementTree` documentation] provides more details. In the examples `{XML}` refers to the same XML structure as in the earlier example.

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

== Tag names ==

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

== Paths ==

Paths are created by combining tag names with a forward slash (/). For example, `parent/child` matches all `child` elements under `parent` element. Notice that if there are multiple `parent` elements that all have `child` elements, `parent/child` xpath will match all these `child` elements.

== Wildcards ==

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

== Current element ==

The current element is denoted with a dot (.). Normally the current element is implicit and does not need to be included in the xpath.

== Parent element ==

The parent element of another element is denoted with two dots (. .). Notice that it is not possible to refer to the parent of the current element.

== Search all sub elements ==

Two forward slashes (//) mean that all sub elements, not only the direct children, are searched. If the search is started from the current element, an explicit dot is required.

== Predicates ==

Predicates allow selecting elements using also other criteria than tag names, for example, attributes or position. They are specified after the normal tag name or path using syntax `path[predicate]`. The path can have

wildcards and other special syntax explained earlier. What predicates the standard `ElementTree` supports is explained in the table below.

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

= Element attributes =

All keywords returning elements, such as *Parse XML*, and *Get Element*, return `ElementTree`'s [<http://docs.python.org/library/xml.etree.elementtree.html#element-objects>]. 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 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 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#truthrules> as in Python].

True examples:

False examples:

Considering `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\\_\(programming\)](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** = '5.0.1'

**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.

Use *Get Element* keyword if you want to get a certain element and not the whole structure. See *Parsing XML* section for more details and examples.

**get\_element** (*source*, *xpath='.'*)

Returns an element in the *source* matching the *xpath*.

The *source* can be a path to an XML file, a string containing XML, or an already parsed XML element. The *xpath* specifies which element to find. See the *introduction* for more details about both the possible sources and the supported xpath syntax.

The keyword fails if more, or less, than one element matches the *xpath*. Use *Get Elements* if you want all matching elements to be returned.

*Parse XML* is recommended for parsing XML when the whole structure is needed. It must be used if there is a need to configure how XML namespaces are handled.

Many other keywords use this keyword internally, and keywords modifying XML are typically documented to both to modify the given source and to return it. Modifying the source does not apply if the source is given as a string. The XML structure parsed based on the string and then modified is nevertheless returned.

**get\_elements** (*source*, *xpath*)

Returns a list of elements in the *source* matching the *xpath*.

The *source* can be a path to an XML file, a string containing XML, or an already parsed XML element. The *xpath* specifies which element to find. See the *introduction* for more details.

Elements matching the *xpath* are returned as a list. If no elements match, an empty list is returned. Use *Get Element* if you want to get exactly one match.

**get\_child\_elements** (*source*, *xpath='.'*)

Returns the child elements of the specified element as a list.

The element whose children to return is specified using *source* and *xpath*. They have exactly the same semantics as with *Get Element* keyword.



All the direct child elements of the specified element are returned. If the element has no children, an empty list is returned.

**get\_element\_count** (*source*, *xpath*='.')

Returns and logs how many elements the given *xpath* matches.

Arguments *source* and *xpath* have exactly the same semantics as with *Get Elements* keyword that this keyword uses internally.

See also *Element Should Exist* and *Element Should Not Exist*.

**element\_should\_exist** (*source*, *xpath*='.', *message*=None)

Verifies that one or more element match the given *xpath*.

Arguments *source* and *xpath* have exactly the same semantics as with *Get Elements* keyword. Keyword passes if the *xpath* matches one or more elements in the *source*. The default error message can be overridden with the *message* argument.

See also *Element Should Not Exist* as well as *Get Element Count* that this keyword uses internally.

**element\_should\_not\_exist** (*source*, *xpath*='.', *message*=None)

Verifies that no element match the given *xpath*.

Arguments *source* and *xpath* have exactly the same semantics as with *Get Elements* keyword. Keyword fails if the *xpath* matches any element in the *source*. The default error message can be overridden with the *message* argument.

See also *Element Should Exist* as well as *Get Element Count* that this keyword uses internally.

**get\_element\_text** (*source*, *xpath*='.', *normalize\_whitespace*=False)

Returns all text of the element, possibly whitespace normalized.

The element whose text to return is specified using *source* and *xpath*. They have exactly the same semantics as with *Get Element* keyword.

This keyword returns all the text of the specified element, including all the text its children and grandchildren contain. If the element has no text, an empty string is returned. The returned text is thus not always the same as the *text* attribute of the element.

By default all whitespace, including newlines and indentation, inside the element is returned as-is. If *normalize\_whitespace* is given a true value (see *Boolean arguments*), then leading and trailing whitespace is stripped, newlines and tabs converted to spaces, and multiple spaces collapsed into one. This is especially useful when dealing with HTML data.

See also *Get Elements Texts*, *Element Text Should Be* and *Element Text Should Match*.

**get\_elements\_texts** (*source*, *xpath*, *normalize\_whitespace*=False)

Returns text of all elements matching *xpath* as a list.

The elements whose text to return is specified using *source* and *xpath*. They have exactly the same semantics as with *Get Elements* keyword.

The text of the matched elements is returned using the same logic as with *Get Element Text*. This includes optional whitespace normalization using the *normalize\_whitespace* option.

**element\_text\_should\_be** (*source*, *expected*, *xpath*='.', *normalize\_whitespace*=False, *message*=None)

Verifies that the text of the specified element is *expected*.

The element whose text is verified is specified using *source* and *xpath*. They have exactly the same semantics as with *Get Element* keyword.

The text to verify is got from the specified element using the same logic as with *Get Element Text*. This includes optional whitespace normalization using the *normalize\_whitespace* option.

The keyword passes if the text of the element is equal to the `expected` value, and otherwise it fails. The default error message can be overridden with the `message` argument. Use *Element Text Should Match* to verify the text against a pattern instead of an exact value.

**element\_text\_should\_match** (*source*, *pattern*, *xpath*='.', *normalize\_whitespace*=False, *message*=None)

Verifies that the text of the specified element matches `expected`.

This keyword works exactly like *Element Text Should Be* except that the `expected` value can be given as a pattern that the text of the element must match.

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

**get\_element\_attribute** (*source*, *name*, *xpath*='.', *default*=None)

Returns the named attribute of the specified element.

The element whose attribute to return is specified using `source` and `xpath`. They have exactly the same semantics as with *Get Element* keyword.

The value of the attribute `name` of the specified element is returned. If the element does not have such element, the `default` value is returned instead.

See also *Get Element Attributes*, *Element Attribute Should Be*, *Element Attribute Should Match* and *Element Should Not Have Attribute*.

**get\_element\_attributes** (*source*, *xpath*='.')

Returns all attributes of the specified element.

The element whose attributes to return is specified using `source` and `xpath`. They have exactly the same semantics as with *Get Element* keyword.

Attributes are returned as a Python dictionary. It is a copy of the original attributes so modifying it has no effect on the XML structure.

Use *Get Element Attribute* to get the value of a single attribute.

**element\_attribute\_should\_be** (*source*, *name*, *expected*, *xpath*='.', *message*=None)

Verifies that the specified attribute is `expected`.

The element whose attribute is verified is specified using `source` and `xpath`. They have exactly the same semantics as with *Get Element* keyword.

The keyword passes if the attribute `name` of the element is equal to the `expected` value, and otherwise it fails. The default error message can be overridden with the `message` argument.

To test that the element does not have a certain attribute, Python `None` (i.e. variable `${NONE}`) can be used as the `expected` value. A cleaner alternative is using *Element Should Not Have Attribute*.

See also *Element Attribute Should Match* and *Get Element Attribute*.

**element\_attribute\_should\_match** (*source*, *name*, *pattern*, *xpath*='.', *message*=None)

Verifies that the specified attribute matches `expected`.

This keyword works exactly like *Element Attribute Should Be* except that the `expected` value can be given as a pattern that the attribute of the element must match.

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

**element\_should\_not\_have\_attribute** (*source*, *name*, *xpath*='.', *message*=None)

Verifies that the specified element does not have attribute `name`.

The element whose attribute is verified is specified using `source` and `xpath`. They have exactly the same semantics as with *Get Element* keyword.

The keyword fails if the specified element has attribute `name`. The default error message can be overridden with the `message` argument.

See also *Get Element Attribute*, *Get Element Attributes*, *Element Text Should Be* and *Element Text Should Match*.

**elements\_should\_be\_equal** (*source*, *expected*, *exclude\_children=False*, *normalize\_whitespace=False*)

Verifies that the given `source` element is equal to `expected`.

Both `source` and `expected` can be given as a path to an XML file, as a string containing XML, or as an already parsed XML element structure. See *introduction* for more information about parsing XML in general.

The keyword passes if the `source` element and `expected` element are equal. This includes testing the tag names, texts, and attributes of the elements. By default also child elements are verified the same way, but this can be disabled by setting `exclude_children` to a true value (see *Boolean arguments*).

All texts inside the given elements are verified, but possible text outside them is not. By default texts must match exactly, but setting `normalize_whitespace` to a true value makes text verification independent on newlines, tabs, and the amount of spaces. For more details about handling text see *Get Element Text* keyword and discussion about elements' *text* and *tail* attributes in the *introduction*.

The last example may look a bit strange because the `<p>` element only has text `Text with`. The reason is that rest of the text inside `<p>` actually belongs to the child elements. This includes the `.` at the end that is the *tail* text of the `<i>` element.

See also *Elements Should Match*.

**elements\_should\_match** (*source*, *expected*, *exclude\_children=False*, *normalize\_whitespace=False*)

Verifies that the given `source` element matches `expected`.

This keyword works exactly like *Elements Should Be Equal* except that texts and attribute values in the `expected` value can be given as patterns.

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

See *Elements Should Be Equal* for more examples.

**set\_element\_tag** (*source*, *tag*, *xpath='.'*)

Sets the tag of the specified element.

The element whose tag to set is specified using `source` and `xpath`. They have exactly the same semantics as with *Get Element* keyword. The resulting XML structure is returned, and if the `source` is an already parsed XML structure, it is also modified in place.

Can only set the tag of a single element. Use *Set Elements Tag* to set the tag of multiple elements in one call.

**set\_elements\_tag** (*source*, *tag*, *xpath='.'*)

Sets the tag of the specified elements.

Like *Set Element Tag* but sets the tag of all elements matching the given `xpath`.

**set\_element\_text** (*source*, *text=None*, *tail=None*, *xpath='.'*)

Sets text and/or tail text of the specified element.

The element whose text to set is specified using `source` and `xpath`. They have exactly the same semantics as with *Get Element* keyword. The resulting XML structure is returned, and if the `source` is an already parsed XML structure, it is also modified in place.

Element's text and tail text are changed only if new `text` and/or `tail` values are given. See *Element attributes* section for more information about *text* and *tail* in general.

Can only set the text/tail of a single element. Use *Set Elements Text* to set the text/tail of multiple elements in one call.

**set\_elements\_text** (*source*, *text*=None, *tail*=None, *xpath*='.')

Sets text and/or tail text of the specified elements.

Like *Set Element Text* but sets the text or tail of all elements matching the given *xpath*.

**set\_element\_attribute** (*source*, *name*, *value*, *xpath*='.')

Sets attribute name of the specified element to *value*.

The element whose attribute to set is specified using *source* and *xpath*. They have exactly the same semantics as with *Get Element* keyword. The resulting XML structure is returned, and if the *source* is an already parsed XML structure, it is also modified in place.

It is possible to both set new attributes and to overwrite existing. Use *Remove Element Attribute* or *Remove Element Attributes* for removing them.

Can only set an attribute of a single element. Use *Set Elements Attribute* to set an attribute of multiple elements in one call.

**set\_elements\_attribute** (*source*, *name*, *value*, *xpath*='.')

Sets attribute name of the specified elements to *value*.

Like *Set Element Attribute* but sets the attribute of all elements matching the given *xpath*.

**remove\_element\_attribute** (*source*, *name*, *xpath*='.')

Removes attribute *name* from the specified element.

The element whose attribute to remove is specified using *source* and *xpath*. They have exactly the same semantics as with *Get Element* keyword. The resulting XML structure is returned, and if the *source* is an already parsed XML structure, it is also modified in place.

It is not a failure to remove a non-existing attribute. Use *Remove Element Attributes* to remove all attributes and *Set Element Attribute* to set them.

Can only remove an attribute from a single element. Use *Remove Elements Attribute* to remove an attribute of multiple elements in one call.

**remove\_elements\_attribute** (*source*, *name*, *xpath*='.')

Removes attribute *name* from the specified elements.

Like *Remove Element Attribute* but removes the attribute of all elements matching the given *xpath*.

**remove\_element\_attributes** (*source*, *xpath*='.')

Removes all attributes from the specified element.

The element whose attributes to remove is specified using *source* and *xpath*. They have exactly the same semantics as with *Get Element* keyword. The resulting XML structure is returned, and if the *source* is an already parsed XML structure, it is also modified in place.

Use *Remove Element Attribute* to remove a single attribute and *Set Element Attribute* to set them.

Can only remove attributes from a single element. Use *Remove Elements Attributes* to remove all attributes of multiple elements in one call.

**remove\_elements\_attributes** (*source*, *xpath*='.')

Removes all attributes from the specified elements.

Like *Remove Element Attributes* but removes all attributes of all elements matching the given *xpath*.

**add\_element** (*source, element, index=None, xpath='.'*)

Adds a child element to the specified element.

The element to whom to add the new element is specified using `source` and `xpath`. They have exactly the same semantics as with *Get Element* keyword. The resulting XML structure is returned, and if the `source` is an already parsed XML structure, it is also modified in place.

The `element` to add can be specified as a path to an XML file or as a string containing XML, or it can be an already parsed XML element. The element is copied before adding so modifying either the original or the added element has no effect on the other. The element is added as the last child by default, but a custom index can be used to alter the position. Indices start from zero (0 = first position, 1 = second position, etc.), and negative numbers refer to positions at the end (-1 = second last position, -2 = third last, etc.).

Use *Remove Element* or *Remove Elements* to remove elements.

**remove\_element** (*source, xpath="", remove\_tail=False*)

Removes the element matching `xpath` from the `source` structure.

The element to remove from the `source` is specified with `xpath` using the same semantics as with *Get Element* keyword. The resulting XML structure is returned, and if the `source` is an already parsed XML structure, it is also modified in place.

The keyword fails if `xpath` does not match exactly one element. Use *Remove Elements* to remove all matched elements.

Element's tail text is not removed by default, but that can be changed by giving `remove_tail` a true value (see *Boolean arguments*). See *Element attributes* section for more information about *tail* in general.

**remove\_elements** (*source, xpath="", remove\_tail=False*)

Removes all elements matching `xpath` from the `source` structure.

The elements to remove from the `source` are specified with `xpath` using the same semantics as with *Get Elements* keyword. The resulting XML structure is returned, and if the `source` is an already parsed XML structure, it is also modified in place.

It is not a failure if `xpath` matches no elements. Use *Remove Element* to remove exactly one element.

Element's tail text is not removed by default, but that can be changed by using `remove_tail` argument similarly as with *Remove Element*.

**clear\_element** (*source, xpath='.', clear\_tail=False*)

Clears the contents of the specified element.

The element to clear is specified using `source` and `xpath`. They have exactly the same semantics as with *Get Element* keyword. The resulting XML structure is returned, and if the `source` is an already parsed XML structure, it is also modified in place.

Clearing the element means removing its text, attributes, and children. Element's tail text is not removed by default, but that can be changed by giving `clear_tail` a true value (see *Boolean arguments*). See *Element attributes* section for more information about *tail* in general.

Use *Remove Element* to remove the whole element.

**copy\_element** (*source, xpath='.'*)

Returns a copy of the specified element.

The element to copy is specified using `source` and `xpath`. They have exactly the same semantics as with *Get Element* keyword.

If the copy or the original element is modified afterwards, the changes have no effect on the other.

**element\_to\_string** (*source*, *xpath*='.', *encoding*=None)

Returns the string representation of the specified element.

The element to convert to a string is specified using *source* and *xpath*. They have exactly the same semantics as with *Get Element* keyword.

By default the string is returned as Unicode. If *encoding* argument is given any value, the string is returned as bytes in the specified encoding. The resulting string never contains the XML declaration.

See also *Log Element* and *Save XML*.

**log\_element** (*source*, *level*='INFO', *xpath*='.')

Logs the string representation of the specified element.

The element specified with *source* and *xpath* is first converted into a string using *Element To String* keyword internally. The resulting string is then logged using the given *level*.

The logged string is also returned.

**save\_xml** (*source*, *path*, *encoding*='UTF-8')

Saves the given element to the specified file.

The element to save is specified with *source* using the same semantics as with *Get Element* keyword.

The file where the element is saved is denoted with *path* and the encoding to use with *encoding*. The resulting file always contains the XML declaration.

The resulting XML file may not be exactly the same as the original: - Comments and processing instructions are always stripped. - Possible doctype and namespace prefixes are only preserved when

*using lxml*.

- Other small differences are possible depending on the ElementTree or lxml version.

Use *Element To String* if you just need a string representation of the element.

**evaluate\_xpath** (*source*, *expression*, *context*='.')

Evaluates the given xpath expression and returns results.

The element in which context the expression is executed is specified using *source* and *context* arguments. They have exactly the same semantics as *source* and *xpath* arguments have with *Get Element* keyword.

The xpath expression to evaluate is given as *expression* argument. The result of the evaluation is returned as-is.

This keyword works only if lxml mode is taken into use when *importing* the library.

**class** robot.libraries.XML.NamespaceStripper (*etree*, *lxml\_etree*=False)

Bases: object

**strip** (*elem*, *preserve*=True, *current\_ns*=None, *top*=True)

**unstrip** (*elem*, *current\_ns*=None, *copied*=False)

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

Bases: object

**find\_all** (*elem*, *xpath*)

**class** robot.libraries.XML.ElementComparator (*comparator*, *normalizer*=None, *exclude\_children*=False)

Bases: object

**compare** (*actual*, *expected*, *location*=None)

```
class robot.libraries.XML.Location (path, is_root=True)
    Bases: object

    child (tag)
```

### robot.libraries.dialogs\_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.

    anchor (anchor=None)
        The anchor value controls how to place the grid within the master when no row/column has any weight.

        The default anchor is nw.

    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 MAXNUMBER/MAXDENOM. Return a tuple of the actual values if no argument is given.

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

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

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

        On Macintosh, XXXXX

        On Unix, there are currently no special attribute values.

    bbox (column=None, row=None, col2=None, row2=None)
        Return a tuple of integer coordinates for the bounding box of this widget controlled by the geometry manager grid.

        If COLUMN, ROW is given the bounding box applies from the cell with row and column 0 to the specified cell. If COL2 and ROW2 are given the bounding box starts at that cell.

        The returned integers specify the offset of the upper left corner in the master widget and the width and height.

    bell (displayof=0)
        Ring a display's bell.
```

**bind** (*sequence=None, func=None, add=None*)

Bind to this widget at event SEQUENCE a call to function FUNC.

SEQUENCE is a string of concatenated event patterns. An event pattern is of the form <MODIFIER-MODIFIER-TYPE-DETAIL> where MODIFIER is one of Control, Mod2, M2, Shift, Mod3, M3, Lock, Mod4, M4, Button1, B1, Mod5, M5 Button2, B2, Meta, M, Button3, B3, Alt, Button4, B4, Double, Button5, B5 Triple, Mod1, M1. TYPE is one of Activate, Enter, Map, ButtonPress, Button, Expose, Motion, ButtonRelease FocusIn, MouseWheel, Circulate, FocusOut, Property, Colormap, Gravity Reparent, Configure, KeyPress, Key, Unmap, Deactivate, KeyRelease Visibility, Destroy, Leave and DETAIL is the button number for ButtonPress, ButtonRelease and DETAIL is the Keysym for KeyPress and KeyRelease. Examples are <Control-Button-1> for pressing Control and mouse button 1 or <Alt-A> for pressing A and the Alt key (KeyPress can be omitted). An event pattern can also be a virtual event of the form <<AS-string>> 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.

**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.

**forget (window)**

The window will be unmapped from the screen and will no longer be managed by wm. toplevel windows will be treated like frame windows once they are no longer managed by wm, however, the menu option configuration will be remembered and the menus will return once the widget is managed again.

**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 (s)****getint (s)****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)**

Set grab for this widget.

A grab directs all events to this and descendant widgets in the application.

**grab\_set\_global** ()

Set global grab for this widget.

A global grab directs all events to this and descendant widgets on the display. Use with caution - other applications do not get events anymore.

**grab\_status** ()

Return None, “local” or “global” if this widget has no, a local or a global grab.

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

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

**grid\_anchor** (*anchor=None*)

The anchor value controls how to place the grid within the master when no row/column has any weight.

The default anchor is nw.

**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.

**iconphoto** (*default=False, \*args*)

Sets the titlebar icon for this window based on the named photo images passed through args. If default is True, this is applied to all future created toplevels as well.

The data in the images is taken as a snapshot at the time of invocation. If the images are later changed, this is not reflected to the titlebar icons. Multiple images are accepted to allow different images sizes to be provided. The window manager may scale provided icons to an appropriate size.

On Windows, the images are packed into a Windows icon structure. This will override an icon specified to `wm_iconbitmap`, and vice versa.

On X, the images are arranged into the `_NET_WM_ICON` X property, which most modern window managers support. An icon specified by `wm_iconbitmap` may exist simultaneously.

On Macintosh, this currently does nothing.

**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.

**manage** (*widget*)

The widget specified will become a stand alone top-level window. The window will be decorated with the window managers title bar, etc.

**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='I'*)

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\_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=False*)

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 is true 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\_forget** (*window*)

The window will be unmapped from the screen and will no longer be managed by wm. toplevel windows will be treated like frame windows once they are no longer managed by wm, however, the menu option configuration will be remembered and the menus will return once the widget is managed again.

**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\_iconphoto** (*default=False, \*args*)

Sets the titlebar icon for this window based on the named photo images passed through args. If default is True, this is applied to all future created toplevels as well.

The data in the images is taken as a snapshot at the time of invocation. If the images are later changed, this is not reflected to the titlebar icons. Multiple images are accepted to allow different images sizes to be

provided. The window manager may scale provided icons to an appropriate size.

On Windows, the images are packed into a Windows icon structure. This will override an icon specified to `wm_iconbitmap`, and vice versa.

On X, the images are arranged into the `_NET_WM_ICON` X property, which most modern window managers support. An icon specified by `wm_iconbitmap` may exist simultaneously.

On Macintosh, this currently does nothing.

**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\_manage** (*widget*)

The widget specified will become a stand alone top-level window. The window will be decorated with the window managers title bar, etc.

**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`.

**anchor** (*anchor=None*)  
 The anchor value controls how to place the grid within the master when no row/column has any weight.

The default anchor is nw.

**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 <<AS-tring>> where AString can be arbitrary. This event can be generated by event\_generate. If events are concatenated they must appear shortly after each other.

FUNC will be called if the event sequence occurs with an instance of Event as argument. If the return value of FUNC is “break” no further bound function is invoked.

An additional boolean parameter ADD specifies whether FUNC will be called additionally to the other bound function or whether it will replace the previous function.

Bind will return an identifier to allow deletion of the bound function with unbind without memory leak.

If FUNC or SEQUENCE is omitted the bound function or list of bound events are returned.

**bind\_all** (*sequence=None, func=None, add=None*)

Bind to all widgets at an event SEQUENCE a call to function FUNC. An additional boolean parameter ADD specifies whether FUNC will be called additionally to the other bound function or whether it will replace the previous function. See bind for the return value.

**bind\_class** (*className, sequence=None, func=None, add=None*)

Bind to widgets with bindtag CLASSNAME at event SEQUENCE a call of function FUNC. An additional boolean parameter ADD specifies whether FUNC will be called additionally to the other bound function or whether it will replace the previous function. See bind for the return value.

**bindtags** (*tagList=None*)

Set or get the list of bindtags for this widget.

With no argument return the list of all bindtags associated with this widget. With a list of strings as argument the bindtags are set to this list. The bindtags determine in which order events are processed (see bind).

**cget** (*key*)

Return the resource value for a KEY given as string.

**client** (*name=None*)

Store NAME in WM\_CLIENT\_MACHINE property of this widget. Return current value.

**clipboard\_append** (*string, \*\*kw*)

Append STRING to the Tk clipboard.

A widget specified at the optional displayof keyword argument specifies the target display. The clipboard can be retrieved with selection\_get.

**clipboard\_clear** (*\*\*kw*)

Clear the data in the Tk clipboard.

A widget specified for the optional displayof keyword argument specifies the target display.

**clipboard\_get** (*\*\*kw*)

Retrieve data from the clipboard on window’s display.

The window keyword defaults to the root window of the Tkinter application.

The type keyword specifies the form in which the data is to be returned and should be an atom name such as STRING or FILE\_NAME. Type defaults to STRING, except on X11, where the default is to try UTF8\_STRING and fall back to STRING.

This command is equivalent to:

selection\_get(CLIPBOARD)

**colormapwindows** (\*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.

**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.

**forget** (*window*)

The window will be unmapped from the screen and will no longer be managed by wm. toplevel windows will be treated like frame windows once they are no longer managed by wm, however, the menu option configuration will be remembered and the menus will return once the widget is managed again.

**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** (*s*)

**getint** (*s*)

**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*)

Set grab for this widget.

A grab directs all events to this and descendant widgets in the application.

**grab\_set\_global** ( )

Set global grab for this widget.

A global grab directs all events to this and descendant widgets on the display. Use with caution - other applications do not get events anymore.



**grab\_status** ()

Return None, “local” or “global” if this widget has no, a local or a global grab.

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

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

**grid\_anchor** (*anchor=None*)

The anchor value controls how to place the grid within the master when no row/column has any weight.

The default anchor is nw.

**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.

**iconphoto** (*default=False, \*args*)

Sets the titlebar icon for this window based on the named photo images passed through args. If default is True, this is applied to all future created toplevels as well.

The data in the images is taken as a snapshot at the time of invocation. If the images are later changed, this is not reflected to the titlebar icons. Multiple images are accepted to allow different images sizes to be provided. The window manager may scale provided icons to an appropriate size.

On Windows, the images are packed into a Windows icon structure. This will override an icon specified to `wm_iconbitmap`, and vice versa.

On X, the images are arranged into the `_NET_WM_ICON` X property, which most modern window managers support. An icon specified by `wm_iconbitmap` may exist simultaneously.

On Macintosh, this currently does nothing.

**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.

**manage** (*widget*)

The widget specified will become a stand alone top-level window. The window will be decorated with the window managers title bar, etc.

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

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

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

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

**nametowidget** (*name*)

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

**option\_add** (*pattern, value, priority=None*)

Set a VALUE (second parameter) for an option PATTERN (first parameter).

An optional third parameter gives the numeric priority (defaults to 80).

**option\_clear** ()

Clear the option database.

It will be reloaded if option\_add is called.

**option\_get** (*name, className*)

Return the value for an option NAME for this widget with CLASSNAME.

Values with higher priority override lower values.

**option\_readfile** (*fileName, priority=None*)

Read file FILENAME into the option database.

An optional second parameter gives the numeric priority.

**overrideredirect** (*boolean=None*)

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

**pack\_propagate** (*flag=['\_noarg\_']*)

Set or get the status for propagation of geometry information.

A boolean argument specifies whether the geometry information of the slaves will determine the size of this widget. If no argument is given the current setting will be returned.

**pack\_slaves** ()

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

**place\_slaves** ()

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

**positionfrom** (*who=None*)

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

**propagate** (*flag=['\_noarg\_']*)

Set or get the status for propagation of geometry information.

A boolean argument specifies whether the geometry information of the slaves will determine the size of this widget. If no argument is given the current setting will be returned.

**protocol** (*name=None, func=None*)

Bind function FUNC to command NAME for this widget. Return the function bound to NAME if None is given. NAME could be e.g. “WM\_SAVE\_YOURSELF” or “WM\_DELETE\_WINDOW”.

**quit** ()

Quit the Tcl interpreter. All widgets will be destroyed.

**register** (*func, subst=None, needcleanup=1*)

Return a newly created Tcl function. If this function is called, the Python function FUNC will be executed. An optional function SUBST can be given which will be executed before FUNC.

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

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

**rowconfigure** (*index*, *cnf*=[], *\*\*kw*)

Configure row INDEX of a grid.

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

**selection\_clear** (*\*\*kw*)

Clear the current X selection.

**selection\_get** (*\*\*kw*)

Return the contents of the current X selection.

A keyword parameter selection specifies the name of the selection and defaults to PRIMARY. A keyword parameter displayof specifies a widget on the display to use. A keyword parameter type specifies the form of data to be fetched, defaulting to STRING except on X11, where UTF8\_STRING is tried before STRING.

**selection\_handle** (*command*, *\*\*kw*)

Specify a function COMMAND to call if the X selection owned by this widget is queried by another application.

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

**selection\_own** (*\*\*kw*)

Become owner of X selection.

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

**selection\_own\_get** (*\*\*kw*)

Return owner of X selection.

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

**send** (*interp*, *cmd*, *\*args*)

Send Tcl command CMD to different interpreter INTERP to be executed.

**setvar** (*name*=*'PY\_VAR'*, *value*=*'1'*)

Set Tcl variable NAME to VALUE.

**show** ()

**size** ()

Return a tuple of the number of column and rows in the grid.

**sizefrom** (*who*=*None*)

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

**slaves** ()

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

**state** (*newstate*=*None*)

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

**title** (*string*=*None*)

Set the title of this widget.

**tk\_bisque()**

Change the color scheme to light brown as used in Tk 3.6 and before.

**tk\_focusFollowsMouse()**

The widget under mouse will get automatically focus. Can not be disabled easily.

**tk\_focusNext()**

Return the next widget in the focus order which follows widget which has currently the focus.

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

**tk\_focusPrev()**

Return previous widget in the focus order. See tk\_focusNext for details.

**tk\_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\_pointery** ()  
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 colormap 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=False*)

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 is true 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 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\_forget** (window)

The window will be unmapped from the screen and will no longer be managed by wm. toplevel windows will be treated like frame windows once they are no longer managed by wm, however, the menu option configuration will be remembered and the menus will return once the widget is managed again.

**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\_iconphoto** (default=False, \*args)

Sets the titlebar icon for this window based on the named photo images passed through args. If default is True, this is applied to all future created toplevels as well.

The data in the images is taken as a snapshot at the time of invocation. If the images are later changed, this is not reflected to the titlebar icons. Multiple images are accepted to allow different images sizes to be provided. The window manager may scale provided icons to an appropriate size.

On Windows, the images are packed into a Windows icon structure. This will override an icon specified to `wm_iconbitmap`, and vice versa.

On X, the images are arranged into the `_NET_WM_ICON` X property, which most modern window managers support. An icon specified by `wm_iconbitmap` may exist simultaneously.

On Macintosh, this currently does nothing.

**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\_manage** (*widget*)

The widget specified will become a stand alone top-level window. The window will be decorated with the window managers title bar, etc.

**wm\_maxsize** (*width=None, height=None*)

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

**wm\_minsize** (*width=None, height=None*)

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

**wm\_overrideredirect** (*boolean=None*)

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

**wm\_positionfrom** (*who=None*)

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

**wm\_protocol** (*name=None, func=None*)

Bind function FUNC to command NAME for this widget. Return the function bound to NAME if None is given. NAME could be e.g. “WM\_SAVE\_YOURSELF” or “WM\_DELETE\_WINDOW”.

**wm\_resizable** (*width=None, height=None*)

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

**wm\_sizefrom** (*who=None*)

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

**wm\_state** (*newstate=None*)

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

**wm\_title** (*string=None*)

Set the title of this widget.

**wm\_transient** (*master=None*)

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

**wm\_withdraw** ()

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

**class** robot.libraries.dialogs\_py.**SelectionDialog** (*message, values*)

Bases: robot.libraries.dialogs\_py.\_TkDialog

**after** (*ms, func=None, \*args*)

Call function once after given time.

MS specifies the time in milliseconds. FUNC gives the function which shall be called. Additional parameters are given as parameters to the function call. Return identifier to cancel scheduling with `after_cancel`.

**after\_cancel** (*id*)

Cancel scheduling of function identified with ID.

Identifier returned by `after` or `after_idle` must be given as first parameter.

**after\_idle** (*func, \*args*)

Call FUNC once if the Tcl main loop has no event to process.

Return an identifier to cancel the scheduling with `after_cancel`.

**anchor** (*anchor=None*)

The anchor value controls how to place the grid within the master when no row/column has any weight.

The default anchor is nw.

**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 <<AS-tring>> where AString can be arbitrary. This event can be generated by event\_generate. If events are concatenated they must appear shortly after each other.

FUNC will be called if the event sequence occurs with an instance of Event as argument. If the return value of FUNC is “break” no further bound function is invoked.

An additional boolean parameter ADD specifies whether FUNC will be called additionally to the other bound function or whether it will replace the previous function.

Bind will return an identifier to allow deletion of the bound function with unbind without memory leak.

If FUNC or SEQUENCE is omitted the bound function or list of bound events are returned.

**bind\_all** (*sequence=None, func=None, add=None*)

Bind to all widgets at an event SEQUENCE a call to function FUNC. An additional boolean parameter ADD specifies whether FUNC will be called additionally to the other bound function or whether it will replace the previous function. See bind for the return value.

**bind\_class** (*className, sequence=None, func=None, add=None*)

Bind to widgets with bindtag CLASSNAME at event SEQUENCE a call of function FUNC. An additional boolean parameter ADD specifies whether FUNC will be called additionally to the other bound function or whether it will replace the previous function. See bind for the return value.

**bindtags** (*tagList=None*)

Set or get the list of bindtags for this widget.

With no argument return the list of all bindtags associated with this widget. With a list of strings as argument the bindtags are set to this list. The bindtags determine in which order events are processed (see bind).

**cget** (*key*)

Return the resource value for a KEY given as string.

**client** (*name=None*)

Store NAME in WM\_CLIENT\_MACHINE property of this widget. Return current value.

**clipboard\_append** (*string, \*\*kw*)

Append STRING to the Tk clipboard.

A widget specified at the optional displayof keyword argument specifies the target display. The clipboard can be retrieved with selection\_get.

**clipboard\_clear** (*\*\*kw*)

Clear the data in the Tk clipboard.

A widget specified for the optional displayof keyword argument specifies the target display.

**clipboard\_get** (*\*\*kw*)

Retrieve data from the clipboard on window’s display.

The window keyword defaults to the root window of the Tkinter application.

The type keyword specifies the form in which the data is to be returned and should be an atom name such as STRING or FILE\_NAME. Type defaults to STRING, except on X11, where the default is to try UTF8\_STRING and fall back to STRING.

This command is equivalent to:

selection\_get(CLIPBOARD)

**colormapwindows** (\*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.

**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.

**forget** (*window*)

The window will be unmapped from the screen and will no longer be managed by wm. toplevel windows will be treated like frame windows once they are no longer managed by wm, however, the menu option configuration will be remembered and the menus will return once the widget is managed again.

**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** (*s*)

**getint** (*s*)

**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*)

Set grab for this widget.

A grab directs all events to this and descendant widgets in the application.

**grab\_set\_global** ( )

Set global grab for this widget.

A global grab directs all events to this and descendant widgets on the display. Use with caution - other applications do not get events anymore.

**grab\_status ()**

Return None, “local” or “global” if this widget has no, a local or a global grab.

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

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

**grid\_anchor** (*anchor=None*)

The anchor value controls how to place the grid within the master when no row/column has any weight.

The default anchor is nw.

**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.

**iconphoto** (*default=False, \*args*)

Sets the titlebar icon for this window based on the named photo images passed through args. If default is True, this is applied to all future created toplevels as well.

The data in the images is taken as a snapshot at the time of invocation. If the images are later changed, this is not reflected to the titlebar icons. Multiple images are accepted to allow different images sizes to be provided. The window manager may scale provided icons to an appropriate size.

On Windows, the images are packed into a Windows icon structure. This will override an icon specified to `wm_iconbitmap`, and vice versa.

On X, the images are arranged into the `_NET_WM_ICON` X property, which most modern window managers support. An icon specified by `wm_iconbitmap` may exist simultaneously.

On Macintosh, this currently does nothing.

**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.

**manage** (*widget*)

The widget specified will become a stand alone top-level window. The window will be decorated with the window managers title bar, etc.

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

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

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

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



**nametowidget** (*name*)

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

**option\_add** (*pattern, value, priority=None*)

Set a VALUE (second parameter) for an option PATTERN (first parameter).

An optional third parameter gives the numeric priority (defaults to 80).

**option\_clear** ()

Clear the option database.

It will be reloaded if option\_add is called.

**option\_get** (*name, className*)

Return the value for an option NAME for this widget with CLASSNAME.

Values with higher priority override lower values.

**option\_readfile** (*fileName, priority=None*)

Read file FILENAME into the option database.

An optional second parameter gives the numeric priority.

**overrideredirect** (*boolean=None*)

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

**pack\_propagate** (*flag=['\_noarg\_']*)

Set or get the status for propagation of geometry information.

A boolean argument specifies whether the geometry information of the slaves will determine the size of this widget. If no argument is given the current setting will be returned.

**pack\_slaves** ()

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

**place\_slaves** ()

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

**positionfrom** (*who=None*)

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

**propagate** (*flag=['\_noarg\_']*)

Set or get the status for propagation of geometry information.

A boolean argument specifies whether the geometry information of the slaves will determine the size of this widget. If no argument is given the current setting will be returned.

**protocol** (*name=None, func=None*)

Bind function FUNC to command NAME for this widget. Return the function bound to NAME if None is given. NAME could be e.g. “WM\_SAVE\_YOURSELF” or “WM\_DELETE\_WINDOW”.

**quit** ()

Quit the Tcl interpreter. All widgets will be destroyed.

**register** (*func, subst=None, needcleanup=1*)

Return a newly created Tcl function. If this function is called, the Python function FUNC will be executed. An optional function SUBST can be given which will be executed before FUNC.

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

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

**rowconfigure** (*index*, *cnf*=[], *\*\*kw*)

Configure row INDEX of a grid.

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

**selection\_clear** (*\*\*kw*)

Clear the current X selection.

**selection\_get** (*\*\*kw*)

Return the contents of the current X selection.

A keyword parameter selection specifies the name of the selection and defaults to PRIMARY. A keyword parameter displayof specifies a widget on the display to use. A keyword parameter type specifies the form of data to be fetched, defaulting to STRING except on X11, where UTF8\_STRING is tried before STRING.

**selection\_handle** (*command*, *\*\*kw*)

Specify a function COMMAND to call if the X selection owned by this widget is queried by another application.

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

**selection\_own** (*\*\*kw*)

Become owner of X selection.

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

**selection\_own\_get** (*\*\*kw*)

Return owner of X selection.

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

**send** (*interp*, *cmd*, *\*args*)

Send Tcl command CMD to different interpreter INTERP to be executed.

**setvar** (*name*=*'PY\_VAR'*, *value*=*'1'*)

Set Tcl variable NAME to VALUE.

**show** ()

**size** ()

Return a tuple of the number of column and rows in the grid.

**sizefrom** (*who*=None)

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

**slaves** ()

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

**state** (*newstate*=None)

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

**title** (*string*=None)

Set the title of this widget.

**tk\_bisque()**

Change the color scheme to light brown as used in Tk 3.6 and before.

**tk\_focusFollowsMouse()**

The widget under mouse will get automatically focus. Can not be disabled easily.

**tk\_focusNext()**

Return the next widget in the focus order which follows widget which has currently the focus.

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

**tk\_focusPrev()**

Return previous widget in the focus order. See tk\_focusNext for details.

**tk\_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\_pointery** ()  
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 colormap 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=False*)

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 is true 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 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\_forget** (window)

The window will be unmapped from the screen and will no longer be managed by wm. toplevel windows will be treated like frame windows once they are no longer managed by wm, however, the menu option configuration will be remembered and the menus will return once the widget is managed again.

**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\_iconphoto** (default=False, \*args)

Sets the titlebar icon for this window based on the named photo images passed through args. If default is True, this is applied to all future created toplevels as well.

The data in the images is taken as a snapshot at the time of invocation. If the images are later changed, this is not reflected to the titlebar icons. Multiple images are accepted to allow different images sizes to be provided. The window manager may scale provided icons to an appropriate size.

On Windows, the images are packed into a Windows icon structure. This will override an icon specified to `wm_iconbitmap`, and vice versa.

On X, the images are arranged into the `_NET_WM_ICON` X property, which most modern window managers support. An icon specified by `wm_iconbitmap` may exist simultaneously.

On Macintosh, this currently does nothing.

**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\_manage** (*widget*)

The widget specified will become a stand alone top-level window. The window will be decorated with the window managers title bar, etc.

**wm\_maxsize** (*width=None, height=None*)

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

**wm\_minsize** (*width=None, height=None*)

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

**wm\_overrideredirect** (*boolean=None*)

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

**wm\_positionfrom** (*who=None*)

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

**wm\_protocol** (*name=None, func=None*)

Bind function FUNC to command NAME for this widget. Return the function bound to NAME if None is given. NAME could be e.g. “WM\_SAVE\_YOURSELF” or “WM\_DELETE\_WINDOW”.

**wm\_resizable** (*width=None, height=None*)

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

**wm\_sizefrom** (*who=None*)

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

**wm\_state** (*newstate=None*)

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

**wm\_title** (*string=None*)

Set the title of this widget.

**wm\_transient** (*master=None*)

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

**wm\_withdraw** ()

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



```
class robot.libraries.dialogs_py.MultipleSelectionDialog (message, values)
```

Bases: robot.libraries.dialogs\_py.\_TkDialog

**after** (*ms, func=None, \*args*)  
 Call function once after given time.

MS specifies the time in milliseconds. FUNC gives the function which shall be called. Additional parameters are given as parameters to the function call. Return identifier to cancel scheduling with after\_cancel.

**after\_cancel** (*id*)  
 Cancel scheduling of function identified with ID.

Identifier returned by after or after\_idle must be given as first parameter.

**after\_idle** (*func, \*args*)  
 Call FUNC once if the Tcl main loop has no event to process.

Return an identifier to cancel the scheduling with after\_cancel.

**anchor** (*anchor=None*)  
 The anchor value controls how to place the grid within the master when no row/column has any weight.

The default anchor is nw.

**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 <<AS-tring>> where AString can be arbitrary. This event can be generated by event\_generate. If events are concatenated they must appear shortly after each other.

FUNC will be called if the event sequence occurs with an instance of Event as argument. If the return value of FUNC is “break” no further bound function is invoked.

An additional boolean parameter ADD specifies whether FUNC will be called additionally to the other bound function or whether it will replace the previous function.

Bind will return an identifier to allow deletion of the bound function with unbind without memory leak.

If FUNC or SEQUENCE is omitted the bound function or list of bound events are returned.

**bind\_all** (*sequence=None, func=None, add=None*)

Bind to all widgets at an event SEQUENCE a call to function FUNC. An additional boolean parameter ADD specifies whether FUNC will be called additionally to the other bound function or whether it will replace the previous function. See bind for the return value.

**bind\_class** (*className, sequence=None, func=None, add=None*)

Bind to widgets with bindtag CLASSNAME at event SEQUENCE a call of function FUNC. An additional boolean parameter ADD specifies whether FUNC will be called additionally to the other bound function or whether it will replace the previous function. See bind for the return value.

**bindtags** (*tagList=None*)

Set or get the list of bindtags for this widget.

With no argument return the list of all bindtags associated with this widget. With a list of strings as argument the bindtags are set to this list. The bindtags determine in which order events are processed (see bind).

**cget** (*key*)

Return the resource value for a KEY given as string.

**client** (*name=None*)

Store NAME in WM\_CLIENT\_MACHINE property of this widget. Return current value.

**clipboard\_append** (*string, \*\*kw*)

Append STRING to the Tk clipboard.

A widget specified at the optional displayof keyword argument specifies the target display. The clipboard can be retrieved with selection\_get.

**clipboard\_clear** (*\*\*kw*)

Clear the data in the Tk clipboard.

A widget specified for the optional displayof keyword argument specifies the target display.

**clipboard\_get** (*\*\*kw*)

Retrieve data from the clipboard on window’s display.

The window keyword defaults to the root window of the Tkinter application.

The type keyword specifies the form in which the data is to be returned and should be an atom name such as STRING or FILE\_NAME. Type defaults to STRING, except on X11, where the default is to try UTF8\_STRING and fall back to STRING.

This command is equivalent to:

selection\_get(CLIPBOARD)

**colormapwindows** (\*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.

**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.

**forget** (*window*)

The window will be unmapped from the screen and will no longer be managed by wm. toplevel windows will be treated like frame windows once they are no longer managed by wm, however, the menu option configuration will be remembered and the menus will return once the widget is managed again.

**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** (*s*)

**getint** (*s*)

**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*)

Set grab for this widget.

A grab directs all events to this and descendant widgets in the application.

**grab\_set\_global** ( )

Set global grab for this widget.

A global grab directs all events to this and descendant widgets on the display. Use with caution - other applications do not get events anymore.

**grab\_status ()**

Return None, “local” or “global” if this widget has no, a local or a global grab.

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

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

**grid\_anchor** (*anchor=None*)

The anchor value controls how to place the grid within the master when no row/column has any weight.

The default anchor is nw.

**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.

**iconphoto** (*default=False, \*args*)

Sets the titlebar icon for this window based on the named photo images passed through args. If default is True, this is applied to all future created toplevels as well.

The data in the images is taken as a snapshot at the time of invocation. If the images are later changed, this is not reflected to the titlebar icons. Multiple images are accepted to allow different images sizes to be provided. The window manager may scale provided icons to an appropriate size.

On Windows, the images are packed into a Windows icon structure. This will override an icon specified to `wm_iconbitmap`, and vice versa.

On X, the images are arranged into the `_NET_WM_ICON` X property, which most modern window managers support. An icon specified by `wm_iconbitmap` may exist simultaneously.

On Macintosh, this currently does nothing.

**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.

**manage** (*widget*)

The widget specified will become a stand alone top-level window. The window will be decorated with the window managers title bar, etc.

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

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

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

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

**nametowidget** (*name*)

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

**option\_add** (*pattern, value, priority=None*)

Set a VALUE (second parameter) for an option PATTERN (first parameter).

An optional third parameter gives the numeric priority (defaults to 80).

**option\_clear** ()

Clear the option database.

It will be reloaded if option\_add is called.

**option\_get** (*name, className*)

Return the value for an option NAME for this widget with CLASSNAME.

Values with higher priority override lower values.

**option\_readfile** (*fileName, priority=None*)

Read file FILENAME into the option database.

An optional second parameter gives the numeric priority.

**overrideredirect** (*boolean=None*)

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

**pack\_propagate** (*flag=['\_noarg\_']*)

Set or get the status for propagation of geometry information.

A boolean argument specifies whether the geometry information of the slaves will determine the size of this widget. If no argument is given the current setting will be returned.

**pack\_slaves** ()

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

**place\_slaves** ()

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

**positionfrom** (*who=None*)

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

**propagate** (*flag=['\_noarg\_']*)

Set or get the status for propagation of geometry information.

A boolean argument specifies whether the geometry information of the slaves will determine the size of this widget. If no argument is given the current setting will be returned.

**protocol** (*name=None, func=None*)

Bind function FUNC to command NAME for this widget. Return the function bound to NAME if None is given. NAME could be e.g. “WM\_SAVE\_YOURSELF” or “WM\_DELETE\_WINDOW”.

**quit** ()

Quit the Tcl interpreter. All widgets will be destroyed.

**register** (*func, subst=None, needcleanup=1*)

Return a newly created Tcl function. If this function is called, the Python function FUNC will be executed. An optional function SUBST can be given which will be executed before FUNC.

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

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

**rowconfigure** (*index*, *cnf*=[], *\*\*kw*)

Configure row INDEX of a grid.

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

**selection\_clear** (*\*\*kw*)

Clear the current X selection.

**selection\_get** (*\*\*kw*)

Return the contents of the current X selection.

A keyword parameter selection specifies the name of the selection and defaults to PRIMARY. A keyword parameter displayof specifies a widget on the display to use. A keyword parameter type specifies the form of data to be fetched, defaulting to STRING except on X11, where UTF8\_STRING is tried before STRING.

**selection\_handle** (*command*, *\*\*kw*)

Specify a function COMMAND to call if the X selection owned by this widget is queried by another application.

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

**selection\_own** (*\*\*kw*)

Become owner of X selection.

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

**selection\_own\_get** (*\*\*kw*)

Return owner of X selection.

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

**send** (*interp*, *cmd*, *\*args*)

Send Tcl command CMD to different interpreter INTERP to be executed.

**setvar** (*name*=*'PY\_VAR'*, *value*=*'1'*)

Set Tcl variable NAME to VALUE.

**show** ()

**size** ()

Return a tuple of the number of column and rows in the grid.

**sizefrom** (*who*=None)

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

**slaves** ()

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

**state** (*newstate*=None)

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

**title** (*string*=None)

Set the title of this widget.



**tk\_bisque()**

Change the color scheme to light brown as used in Tk 3.6 and before.

**tk\_focusFollowsMouse()**

The widget under mouse will get automatically focus. Can not be disabled easily.

**tk\_focusNext()**

Return the next widget in the focus order which follows widget which has currently the focus.

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

**tk\_focusPrev()**

Return previous widget in the focus order. See tk\_focusNext for details.

**tk\_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\_pointery** ()  
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 colormap 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=False*)

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 is true 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 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\_forget** (window)

The window will be unmapped from the screen and will no longer be managed by wm. toplevel windows will be treated like frame windows once they are no longer managed by wm, however, the menu option configuration will be remembered and the menus will return once the widget is managed again.

**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\_iconphoto** (default=False, \*args)

Sets the titlebar icon for this window based on the named photo images passed through args. If default is True, this is applied to all future created toplevels as well.

The data in the images is taken as a snapshot at the time of invocation. If the images are later changed, this is not reflected to the titlebar icons. Multiple images are accepted to allow different images sizes to be provided. The window manager may scale provided icons to an appropriate size.

On Windows, the images are packed into a Windows icon structure. This will override an icon specified to `wm_iconbitmap`, and vice versa.

On X, the images are arranged into the `_NET_WM_ICON` X property, which most modern window managers support. An icon specified by `wm_iconbitmap` may exist simultaneously.

On Macintosh, this currently does nothing.

**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\_manage** (*widget*)

The widget specified will become a stand alone top-level window. The window will be decorated with the window managers title bar, etc.

**wm\_maxsize** (*width=None, height=None*)

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

**wm\_minsize** (*width=None, height=None*)

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

**wm\_overrideredirect** (*boolean=None*)

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

**wm\_positionfrom** (*who=None*)

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

**wm\_protocol** (*name=None, func=None*)

Bind function FUNC to command NAME for this widget. Return the function bound to NAME if None is given. NAME could be e.g. “WM\_SAVE\_YOURSELF” or “WM\_DELETE\_WINDOW”.

**wm\_resizable** (*width=None, height=None*)

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

**wm\_sizefrom** (*who=None*)

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

**wm\_state** (*newstate=None*)

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

**wm\_title** (*string=None*)

Set the title of this widget.

**wm\_transient** (*master=None*)

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

**wm\_withdraw** ()

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

---

```
class robot.libraries.dialogs_py.PassFailDialog (message, value=None, **extra)
```

Bases: robot.libraries.dialogs\_py.\_TkDialog

**after** (*ms, func=None, \*args*)  
 Call function once after given time.

MS specifies the time in milliseconds. FUNC gives the function which shall be called. Additional parameters are given as parameters to the function call. Return identifier to cancel scheduling with after\_cancel.

**after\_cancel** (*id*)  
 Cancel scheduling of function identified with ID.

Identifier returned by after or after\_idle must be given as first parameter.

**after\_idle** (*func, \*args*)  
 Call FUNC once if the Tcl main loop has no event to process.

Return an identifier to cancel the scheduling with after\_cancel.

**anchor** (*anchor=None*)  
 The anchor value controls how to place the grid within the master when no row/column has any weight.

The default anchor is nw.

**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 <<AS-tring>> where AString can be arbitrary. This event can be generated by event\_generate. If events are concatenated they must appear shortly after each other.

FUNC will be called if the event sequence occurs with an instance of Event as argument. If the return value of FUNC is “break” no further bound function is invoked.

An additional boolean parameter ADD specifies whether FUNC will be called additionally to the other bound function or whether it will replace the previous function.

Bind will return an identifier to allow deletion of the bound function with unbind without memory leak.

If FUNC or SEQUENCE is omitted the bound function or list of bound events are returned.

**bind\_all** (*sequence=None, func=None, add=None*)

Bind to all widgets at an event SEQUENCE a call to function FUNC. An additional boolean parameter ADD specifies whether FUNC will be called additionally to the other bound function or whether it will replace the previous function. See bind for the return value.

**bind\_class** (*className, sequence=None, func=None, add=None*)

Bind to widgets with bindtag CLASSNAME at event SEQUENCE a call of function FUNC. An additional boolean parameter ADD specifies whether FUNC will be called additionally to the other bound function or whether it will replace the previous function. See bind for the return value.

**bindtags** (*tagList=None*)

Set or get the list of bindtags for this widget.

With no argument return the list of all bindtags associated with this widget. With a list of strings as argument the bindtags are set to this list. The bindtags determine in which order events are processed (see bind).

**cget** (*key*)

Return the resource value for a KEY given as string.

**client** (*name=None*)

Store NAME in WM\_CLIENT\_MACHINE property of this widget. Return current value.

**clipboard\_append** (*string, \*\*kw*)

Append STRING to the Tk clipboard.

A widget specified at the optional displayof keyword argument specifies the target display. The clipboard can be retrieved with selection\_get.

**clipboard\_clear** (*\*\*kw*)

Clear the data in the Tk clipboard.

A widget specified for the optional displayof keyword argument specifies the target display.

**clipboard\_get** (*\*\*kw*)

Retrieve data from the clipboard on window’s display.

The window keyword defaults to the root window of the Tkinter application.

The type keyword specifies the form in which the data is to be returned and should be an atom name such as STRING or FILE\_NAME. Type defaults to STRING, except on X11, where the default is to try UTF8\_STRING and fall back to STRING.

This command is equivalent to:

selection\_get(CLIPBOARD)



**colormapwindows** (\*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.

**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.

**forget** (*window*)

The window will be unmapped from the screen and will no longer be managed by wm. toplevel windows will be treated like frame windows once they are no longer managed by wm, however, the menu option configuration will be remembered and the menus will return once the widget is managed again.

**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** (*s*)

**getint** (*s*)

**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*)

Set grab for this widget.

A grab directs all events to this and descendant widgets in the application.

**grab\_set\_global** ( )

Set global grab for this widget.

A global grab directs all events to this and descendant widgets on the display. Use with caution - other applications do not get events anymore.

**grab\_status ()**

Return None, “local” or “global” if this widget has no, a local or a global grab.

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

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

**grid\_anchor** (*anchor=None*)

The anchor value controls how to place the grid within the master when no row/column has any weight.

The default anchor is nw.

**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.

**iconphoto** (*default=False, \*args*)

Sets the titlebar icon for this window based on the named photo images passed through args. If default is True, this is applied to all future created toplevels as well.

The data in the images is taken as a snapshot at the time of invocation. If the images are later changed, this is not reflected to the titlebar icons. Multiple images are accepted to allow different images sizes to be provided. The window manager may scale provided icons to an appropriate size.

On Windows, the images are packed into a Windows icon structure. This will override an icon specified to `wm_iconbitmap`, and vice versa.

On X, the images are arranged into the `_NET_WM_ICON` X property, which most modern window managers support. An icon specified by `wm_iconbitmap` may exist simultaneously.

On Macintosh, this currently does nothing.

**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.

**manage** (*widget*)

The widget specified will become a stand alone top-level window. The window will be decorated with the window managers title bar, etc.

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

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

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

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

**nametowidget** (*name*)

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

**option\_add** (*pattern, value, priority=None*)

Set a VALUE (second parameter) for an option PATTERN (first parameter).

An optional third parameter gives the numeric priority (defaults to 80).

**option\_clear** ()

Clear the option database.

It will be reloaded if option\_add is called.

**option\_get** (*name, className*)

Return the value for an option NAME for this widget with CLASSNAME.

Values with higher priority override lower values.

**option\_readfile** (*fileName, priority=None*)

Read file FILENAME into the option database.

An optional second parameter gives the numeric priority.

**overrideredirect** (*boolean=None*)

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

**pack\_propagate** (*flag=['\_noarg\_']*)

Set or get the status for propagation of geometry information.

A boolean argument specifies whether the geometry information of the slaves will determine the size of this widget. If no argument is given the current setting will be returned.

**pack\_slaves** ()

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

**place\_slaves** ()

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

**positionfrom** (*who=None*)

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

**propagate** (*flag=['\_noarg\_']*)

Set or get the status for propagation of geometry information.

A boolean argument specifies whether the geometry information of the slaves will determine the size of this widget. If no argument is given the current setting will be returned.

**protocol** (*name=None, func=None*)

Bind function FUNC to command NAME for this widget. Return the function bound to NAME if None is given. NAME could be e.g. “WM\_SAVE\_YOURSELF” or “WM\_DELETE\_WINDOW”.

**quit** ()

Quit the Tcl interpreter. All widgets will be destroyed.

**register** (*func, subst=None, needcleanup=1*)

Return a newly created Tcl function. If this function is called, the Python function FUNC will be executed. An optional function SUBST can be given which will be executed before FUNC.

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

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

**rowconfigure** (*index*, *cnf*=[], *\*\*kw*)

Configure row INDEX of a grid.

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

**selection\_clear** (*\*\*kw*)

Clear the current X selection.

**selection\_get** (*\*\*kw*)

Return the contents of the current X selection.

A keyword parameter selection specifies the name of the selection and defaults to PRIMARY. A keyword parameter displayof specifies a widget on the display to use. A keyword parameter type specifies the form of data to be fetched, defaulting to STRING except on X11, where UTF8\_STRING is tried before STRING.

**selection\_handle** (*command*, *\*\*kw*)

Specify a function COMMAND to call if the X selection owned by this widget is queried by another application.

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

**selection\_own** (*\*\*kw*)

Become owner of X selection.

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

**selection\_own\_get** (*\*\*kw*)

Return owner of X selection.

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

**send** (*interp*, *cmd*, *\*args*)

Send Tcl command CMD to different interpreter INTERP to be executed.

**setvar** (*name*=*'PY\_VAR'*, *value*=*'1'*)

Set Tcl variable NAME to VALUE.

**show** ()

**size** ()

Return a tuple of the number of column and rows in the grid.

**sizefrom** (*who*=None)

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

**slaves** ()

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

**state** (*newstate*=None)

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

**title** (*string*=None)

Set the title of this widget.

**tk\_bisque()**

Change the color scheme to light brown as used in Tk 3.6 and before.

**tk\_focusFollowsMouse()**

The widget under mouse will get automatically focus. Can not be disabled easily.

**tk\_focusNext()**

Return the next widget in the focus order which follows widget which has currently the focus.

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

**tk\_focusPrev()**

Return previous widget in the focus order. See tk\_focusNext for details.

**tk\_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\_pointery** ()  
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 colormap 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=False*)

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 is true 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 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\_forget** (window)

The window will be unmapped from the screen and will no longer be managed by wm. toplevel windows will be treated like frame windows once they are no longer managed by wm, however, the menu option configuration will be remembered and the menus will return once the widget is managed again.

**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\_iconphoto** (default=False, \*args)

Sets the titlebar icon for this window based on the named photo images passed through args. If default is True, this is applied to all future created toplevels as well.

The data in the images is taken as a snapshot at the time of invocation. If the images are later changed, this is not reflected to the titlebar icons. Multiple images are accepted to allow different images sizes to be provided. The window manager may scale provided icons to an appropriate size.

On Windows, the images are packed into a Windows icon structure. This will override an icon specified to `wm_iconbitmap`, and vice versa.

On X, the images are arranged into the `_NET_WM_ICON` X property, which most modern window managers support. An icon specified by `wm_iconbitmap` may exist simultaneously.

On Macintosh, this currently does nothing.

**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\_manage** (*widget*)

The widget specified will become a stand alone top-level window. The window will be decorated with the window managers title bar, etc.

**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.body module

```
class robot.model.body.BodyItem
    Bases: robot.model.modelobject.ModelObject

    KEYWORD = 'KEYWORD'
    SETUP = 'SETUP'
    TEARDOWN = 'TEARDOWN'
    FOR = 'FOR'
    ITERATION = 'ITERATION'
    IF_ELSE_ROOT = 'IF/ELSE ROOT'
    IF = 'IF'
    ELSE_IF = 'ELSE IF'
    ELSE = 'ELSE'
    TRY_EXCEPT_ROOT = 'TRY/EXCEPT ROOT'
    TRY = 'TRY'
    EXCEPT = 'EXCEPT'
    FINALLY = 'FINALLY'
    WHILE = 'WHILE'
    RETURN = 'RETURN'
    CONTINUE = 'CONTINUE'
    BREAK = 'BREAK'
    MESSAGE = 'MESSAGE'
    type = None
    id
        Item id in format like s1-t3-k1.
        See TestSuite.id for more information.
    has_setup
    has_teardown
```

**config** (*\*\*attributes*)

Configure model object with given attributes.

`obj.config(name='Example', doc='Something')` is equivalent to setting `obj.name = 'Example'` and `obj.doc = 'Something'`.

New in Robot Framework 4.0.

**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.

**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.

**parent**

**repr\_args** = ()

**class** `robot.model.body.BaseBody` (*parent=None, items=None*)

Bases: [`robot.model.itemlist.ItemList`](#)

Base class for Body and Branches objects.

**keyword\_class** = None

**for\_class** = None

**if\_class** = None

**try\_class** = None

**while\_class** = None

**return\_class** = None

**continue\_class** = None

**break\_class** = None

**message\_class** = None

**classmethod** **register** (*item\_class*)

Register a virtual subclass of an ABC.

Returns the subclass, to allow usage as a class decorator.

**create**

**create\_keyword** (*\*args, \*\*kwargs*)

**create\_for** (*\*args, \*\*kwargs*)

**create\_if** (*\*args, \*\*kwargs*)

**create\_try** (*\*args, \*\*kwargs*)

**create\_while** (\*args, \*\*kwargs)

**create\_return** (\*args, \*\*kwargs)

**create\_continue** (\*args, \*\*kwargs)

**create\_break** (\*args, \*\*kwargs)

**create\_message** (\*args, \*\*kwargs)

**filter** (keywords=None, messages=None, predicate=None)

Filter body items based on type and/or custom predicate.

To include or exclude items based on types, give matching arguments True or False values. For example, to include only keywords, use `body.filter(keywords=True)` and to exclude messages use `body.filter(messages=False)`. Including and excluding by types at the same time is not supported and filtering my messages is supported only if the Body object actually supports messages.

Custom `predicate` is a callable getting each body item as an argument that must return True/False depending on should the item be included or not.

Selected items are returned as a list and the original body is not modified.

It was earlier possible to filter also based on FOR and IF types. That support was removed in RF 5.0 because it was not considered useful in general and because adding support for all new control structures would have required extra work. To exclude all control structures, use `body.filter(keywords=True, messages=True)` and to only include them use `body.filter(keywords=False, messages=False)`. For more detailed filtering it is possible to use `predicate`.

**flatten** ()

Return steps so that IF and TRY structures are flattened.

Basically the IF/ELSE and TRY/EXCEPT root elements are replaced with their branches. This is how they are shown in log files.

**append** (item)

S.append(value) – append value to the end of the sequence

**clear** () → None – remove all items from S

**count** (value) → integer – return number of occurrences of value

**extend** (items)

S.extend(iterable) – extend sequence by appending elements from the iterable

**index** (value[, start[, stop]]) → integer – return first index of value.

Raises ValueError if the value is not present.

Supporting start and stop arguments is optional, but recommended.

**insert** (index, item)

S.insert(index, value) – insert value before index

**pop** ([index]) → item – remove and return item at index (default last).

Raise IndexError if list is empty or index is out of range.

**remove** (value)

S.remove(value) – remove first occurrence of value. Raise ValueError if the value is not present.

**reverse** ()

S.reverse() – reverse *IN PLACE*

**sort** ()

**visit** (*visitor*)

**class** `robot.model.body.Body` (*parent=None, items=None*)

Bases: `robot.model.body.BaseBody`

A list-like object representing body of a suite, a test or a keyword.

Body contains the keywords and other structures such as FOR loops.

**append** (*item*)

S.append(value) – append value to the end of the sequence

**break\_class**

alias of `robot.model.control.Break`

**clear** () → None – remove all items from S

**continue\_class**

alias of `robot.model.control.Continue`

**count** (*value*) → integer – return number of occurrences of value

**create**

**create\_break** (\*args, \*\*kwargs)

**create\_continue** (\*args, \*\*kwargs)

**create\_for** (\*args, \*\*kwargs)

**create\_if** (\*args, \*\*kwargs)

**create\_keyword** (\*args, \*\*kwargs)

**create\_message** (\*args, \*\*kwargs)

**create\_return** (\*args, \*\*kwargs)

**create\_try** (\*args, \*\*kwargs)

**create\_while** (\*args, \*\*kwargs)

**extend** (*items*)

S.extend(iterable) – extend sequence by appending elements from the iterable

**filter** (*keywords=None, messages=None, predicate=None*)

Filter body items based on type and/or custom predicate.

To include or exclude items based on types, give matching arguments True or False values. For example, to include only keywords, use `body.filter(keywords=True)` and to exclude messages use `body.filter(messages=False)`. Including and excluding by types at the same time is not supported and filtering my messages is supported only if the Body object actually supports messages.

Custom *predicate* is a callable getting each body item as an argument that must return True/False depending on should the item be included or not.

Selected items are returned as a list and the original body is not modified.

It was earlier possible to filter also based on FOR and IF types. That support was removed in RF 5.0 because it was not considered useful in general and because adding support for all new control structures would have required extra work. To exclude all control structures, use `body.filter(keywords=True, messages=True)` and to only include them use `body.filter(keywords=False, messages=False)`. For more detailed filtering it is possible to use *predicate*.



**flatten()**  
 Return steps so that IF and TRY structures are flattened.  
 Basically the IF/ELSE and TRY/EXCEPT root elements are replaced with their branches. This is how they are shown in log files.

**for\_class**  
 alias of `robot.model.control.For`

**if\_class**  
 alias of `robot.model.control.If`

**index** (*value* [, *start* [, *stop* ]]) → integer – return first index of value.  
 Raises ValueError if the value is not present.  
 Supporting start and stop arguments is optional, but recommended.

**insert** (*index*, *item*)  
 S.insert(*index*, *value*) – insert value before index

**keyword\_class**  
 alias of `robot.model.keyword.Keyword`

**message\_class = None**

**pop** ([*index*]) → item – remove and return item at index (default last).  
 Raise IndexError if list is empty or index is out of range.

**classmethod register** (*item\_class*)  
 Register a virtual subclass of an ABC.  
 Returns the subclass, to allow usage as a class decorator.

**remove** (*value*)  
 S.remove(*value*) – remove first occurrence of value. Raise ValueError if the value is not present.

**return\_class**  
 alias of `robot.model.control.Return`

**reverse()**  
 S.reverse() – reverse *IN PLACE*

**sort()**

**try\_class**  
 alias of `robot.model.control.Try`

**visit** (*visitor*)

**while\_class**  
 alias of `robot.model.control.While`

**class** `robot.model.body.Branches` (*branch\_class*, *parent=None*, *items=None*)  
 Bases: `robot.model.body.BaseBody`  
 A list-like object representing branches IF and TRY objects contain.

**branch\_class**

**create\_branch** (\**args*, \*\**kwargs*)

**append** (*item*)  
 S.append(*value*) – append value to the end of the sequence

**break\_class = None**

**clear** () → None – remove all items from S

**continue\_class** = None

**count** (value) → integer – return number of occurrences of value

**create**

**create\_break** (\*args, \*\*kwargs)

**create\_continue** (\*args, \*\*kwargs)

**create\_for** (\*args, \*\*kwargs)

**create\_if** (\*args, \*\*kwargs)

**create\_keyword** (\*args, \*\*kwargs)

**create\_message** (\*args, \*\*kwargs)

**create\_return** (\*args, \*\*kwargs)

**create\_try** (\*args, \*\*kwargs)

**create\_while** (\*args, \*\*kwargs)

**extend** (items)

S.extend(iterable) – extend sequence by appending elements from the iterable

**filter** (keywords=None, messages=None, predicate=None)

Filter body items based on type and/or custom predicate.

To include or exclude items based on types, give matching arguments True or False values. For example, to include only keywords, use `body.filter(keywords=True)` and to exclude messages use `body.filter(messages=False)`. Including and excluding by types at the same time is not supported and filtering my messages is supported only if the Body object actually supports messages.

Custom predicate is a callable getting each body item as an argument that must return True/False depending on should the item be included or not.

Selected items are returned as a list and the original body is not modified.

It was earlier possible to filter also based on FOR and IF types. That support was removed in RF 5.0 because it was not considered useful in general and because adding support for all new control structures would have required extra work. To exclude all control structures, use `body.filter(keywords=True, messages=True)` and to only include them use `body.filter(keywords=False, messages=False)`. For more detailed filtering it is possible to use predicate.

**flatten** ()

Return steps so that IF and TRY structures are flattened.

Basically the IF/ELSE and TRY/EXCEPT root elements are replaced with their branches. This is how they are shown in log files.

**for\_class** = None

**if\_class** = None

**index** (value[, start[, stop]]) → integer – return first index of value.

Raises ValueError if the value is not present.

Supporting start and stop arguments is optional, but recommended.

**insert** (index, item)

S.insert(index, value) – insert value before index

```

keyword_class = None
message_class = None
pop ([index]) → item – remove and return item at index (default last).
    Raise IndexError if list is empty or index is out of range.
classmethod register (item_class)
    Register a virtual subclass of an ABC.

    Returns the subclass, to allow usage as a class decorator.
remove (value)
    S.remove(value) – remove first occurrence of value. Raise ValueError if the value is not present.
return_class = None
reverse ()
    S.reverse() – reverse IN PLACE
sort ()
try_class = None
visit (visitor)
while_class = None

```

### robot.model.configurer module

```

class robot.model.configurer.SuiteConfigurer (name=None, doc=None, meta-
                                              data=None, set_tags=None, in-
                                              clude_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 suites.

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

**end\_body\_item** (*item*)

Called, by default, when keywords, messages or control structures end.

More specific `end_keyword()`, `end_message()`, `:meth:'end_for'`, etc. can be implemented to visit only keywords, messages or specific control structures.

Default implementation does nothing.

**end\_break** (*break\_*)

Called when a BREAK element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_continue** (*continue\_*)

Called when a CONTINUE element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_for** (*for\_*)

Called when a FOR loop ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_for\_iteration** (*iteration*)

Called when a FOR loop iteration ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_if** (*if\_*)

Called when an IF/ELSE structure ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_if\_branch** (*branch*)

Called when an IF/ELSE branch ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_keyword** (*keyword*)

Called when a keyword ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_message** (*msg*)

Called when a message ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_return** (*return\_*)

Called when a RETURN element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_suite** (*suite*)

Called when a suite ends. Default implementation does nothing.

**end\_test** (*test*)

Called when a test ends. Default implementation does nothing.

**end\_try** (*try\_*)

Called when a TRY/EXCEPT structure ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_try\_branch** (*branch*)

Called when TRY, EXCEPT, ELSE and FINALLY branches end.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_while** (*while\_*)

Called when a WHILE loop ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_while\_iteration** (*iteration*)

Called when a WHILE loop iteration ends.

By default, calls `end_body_item()` which, by default, does nothing.

**start\_body\_item** (*item*)

Called, by default, when keywords, messages or control structures start.

More specific `start_keyword()`, `start_message()`, `:meth:'start_for`, etc. can be implemented to visit only keywords, messages or specific control structures.

Can return explicit `False` to stop visiting. Default implementation does nothing.

**start\_break** (*break\_*)

Called when a BREAK element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_continue** (*continue\_*)

Called when a CONTINUE element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_for** (*for\_*)

Called when a FOR loop starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_for\_iteration** (*iteration*)

Called when a FOR loop iteration starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_if** (*if\_*)

Called when an IF/ELSE structure starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_if\_branch** (*branch*)

Called when an IF/ELSE branch starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_keyword** (*keyword*)

Called when a keyword starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_message** (*msg*)

Called when a message starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_return** (*return\_*)

Called when a RETURN element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_suite** (*suite*)

Called when a suite starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

**start\_test** (*test*)

Called when a test starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

**start\_try** (*try\_*)

Called when a TRY/EXCEPT structure starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_try\_branch** (*branch*)

Called when TRY, EXCEPT, ELSE or FINALLY branches start.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_while** (*while\_*)

Called when a WHILE loop starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_while\_iteration** (*iteration*)

Called when a WHILE loop iteration starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**visit\_break** (*break\_*)

Visits BREAK elements.

**visit\_continue** (*continue\_*)

Visits CONTINUE elements.

**visit\_for** (*for\_*)

Implements traversing through FOR loops.

Can be overridden to allow modifying the passed in `for_` without calling `start_for()` or `end_for()` nor visiting body.

**visit\_for\_iteration** (*iteration*)

Implements traversing through single FOR loop iteration.

This is only used with the result side model because on the running side there are no iterations.

Can be overridden to allow modifying the passed in `iteration` without calling `start_for_iteration()` or `end_for_iteration()` nor visiting body.

**visit\_if** (*if\_*)

Implements traversing through IF/ELSE structures.

Notice that `if_` does not have any data directly. Actual IF/ELSE branches are in its body and visited using `visit_if_branch()`.

Can be overridden to allow modifying the passed in `if_` without calling `start_if()` or `end_if()` nor visiting branches.

**visit\_if\_branch** (*branch*)

Implements traversing through single IF/ELSE branch.

Can be overridden to allow modifying the passed in *branch* without calling `start_if_branch()` or `end_if_branch()` nor visiting body.

**visit\_keyword** (*kw*)

Implements traversing through keywords.

Can be overridden to allow modifying the passed in *kw* without calling `start_keyword()` or `end_keyword()` nor visiting the body of the keyword

**visit\_message** (*msg*)

Implements visiting messages.

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

**visit\_return** (*return\_*)

Visits a RETURN elements.

**visit\_test** (*test*)

Implements traversing through tests.

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

**visit\_try** (*try\_*)

Implements traversing through TRY/EXCEPT structures.

This method is used with the TRY/EXCEPT root element. Actual TRY, EXCEPT, ELSE and FINALLY branches are visited separately using `visit_try_branch()`.

**visit\_try\_branch** (*branch*)

Visits individual TRY, EXCEPT, ELSE and FINALLY branches.

**visit\_while** (*while\_*)

Implements traversing through WHILE loops.

Can be overridden to allow modifying the passed in *while\_* without calling `start_while()` or `end_while()` nor visiting body.

**visit\_while\_iteration** (*iteration*)

Implements traversing through single WHILE loop iteration.

This is only used with the result side model because on the running side there are no iterations.

Can be overridden to allow modifying the passed in *iteration* without calling `start_while_iteration()` or `end_while_iteration()` nor visiting body.

## robot.model.control module

```
class robot.model.control.For (variables=(), flavor='IN', values=(), parent=None)
```

Bases: `robot.model.body.BodyItem`

```
type = 'FOR'
```

```
body_class
```

alias of `robot.model.body.Body`

```
repr_args = ('variables', 'flavor', 'values')
```

```
variables
```

**flavor**

**values**

**parent**

**body**

**keywords**

Deprecated since Robot Framework 4.0. Use *body* instead.

**visit** (*visitor*)

**BREAK** = 'BREAK'

**CONTINUE** = 'CONTINUE'

**ELSE** = 'ELSE'

**ELSE\_IF** = 'ELSE IF'

**EXCEPT** = 'EXCEPT'

**FINALLY** = 'FINALLY'

**FOR** = 'FOR'

**IF** = 'IF'

**IF\_ELSE\_ROOT** = 'IF/ELSE ROOT'

**ITERATION** = 'ITERATION'

**KEYWORD** = 'KEYWORD'

**MESSAGE** = 'MESSAGE'

**RETURN** = 'RETURN'

**SETUP** = 'SETUP'

**TEARDOWN** = 'TEARDOWN'

**TRY** = 'TRY'

**TRY\_EXCEPT\_ROOT** = 'TRY/EXCEPT ROOT'

**WHILE** = 'WHILE'

**config** (*\*\*attributes*)

Configure model object with given attributes.

`obj.config(name='Example', doc='Something')` is equivalent to setting `obj.name = 'Example'` and `obj.doc = 'Something'`.

New in Robot Framework 4.0.

**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.

**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.

**has\_setup**

**has\_teardown**

**id**

Item id in format like s1-t3-k1.

See `TestSuite.id` for more information.

**class** `robot.model.control.While` (*condition=None, limit=None, parent=None*)

Bases: `robot.model.body.BodyItem`

**type** = 'WHILE'

**body\_class**

alias of `robot.model.body.Body`

**repr\_args** = ('condition', 'limit')

**condition**

**limit**

**parent**

**body**

**visit** (*visitor*)

**BREAK** = 'BREAK'

**CONTINUE** = 'CONTINUE'

**ELSE** = 'ELSE'

**ELSE\_IF** = 'ELSE IF'

**EXCEPT** = 'EXCEPT'

**FINALLY** = 'FINALLY'

**FOR** = 'FOR'

**IF** = 'IF'

**IF\_ELSE\_ROOT** = 'IF/ELSE ROOT'

**ITERATION** = 'ITERATION'

**KEYWORD** = 'KEYWORD'

**MESSAGE** = 'MESSAGE'

**RETURN** = 'RETURN'

**SETUP** = 'SETUP'

**TEARDOWN** = 'TEARDOWN'

**TRY** = 'TRY'

**TRY\_EXCEPT\_ROOT** = 'TRY/EXCEPT ROOT'

**WHILE** = 'WHILE'

**config** (\*\*attributes)

Configure model object with given attributes.

`obj.config(name='Example', doc='Something')` is equivalent to setting `obj.name = 'Example'` and `obj.doc = 'Something'`.

New in Robot Framework 4.0.

**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.

**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.

**has\_setup**

**has\_teardown**

**id**

Item id in format like `s1-t3-k1`.

See `TestSuite.id` for more information.

**class** `robot.model.control.IfBranch` (type='IF', condition=None, parent=None)

Bases: `robot.model.body.BodyItem`

**body\_class**

alias of `robot.model.body.Body`

**repr\_args** = ('type', 'condition')

**type**

**condition**

**parent**

**body**

**id**

Branch id omits IF/ELSE root from the parent id part.

**visit** (visitor)

**BREAK** = 'BREAK'

**CONTINUE** = 'CONTINUE'

**ELSE** = 'ELSE'

**ELSE\_IF** = 'ELSE IF'

**EXCEPT** = 'EXCEPT'

**FINALLY** = 'FINALLY'

```

FOR = 'FOR'
IF = 'IF'
IF_ELSE_ROOT = 'IF/ELSE ROOT'
ITERATION = 'ITERATION'
KEYWORD = 'KEYWORD'
MESSAGE = 'MESSAGE'
RETURN = 'RETURN'
SETUP = 'SETUP'
TEARDOWN = 'TEARDOWN'
TRY = 'TRY'
TRY_EXCEPT_ROOT = 'TRY/EXCEPT ROOT'
WHILE = 'WHILE'

config (**attributes)
    Configure model object with given attributes.

    obj.config(name='Example', doc='Something') is equivalent to setting obj.name =
    'Example' and obj.doc = 'Something'.

    New in Robot Framework 4.0.

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.

deepcopy (**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.

has_setup
has_teardown

class robot.model.control.If(parent=None)
    Bases: robot.model.body.BodyItem

    IF/ELSE structure root. Branches are stored in body.

    type = 'IF/ELSE ROOT'

    branch_class
        alias of IfBranch

    branches_class
        alias of robot.model.body.Branches

    parent

```

**body**

**id**  
Root IF/ELSE id is always None.

**visit** (*visitor*)

**BREAK** = 'BREAK'

**CONTINUE** = 'CONTINUE'

**ELSE** = 'ELSE'

**ELSE\_IF** = 'ELSE IF'

**EXCEPT** = 'EXCEPT'

**FINALLY** = 'FINALLY'

**FOR** = 'FOR'

**IF** = 'IF'

**IF\_ELSE\_ROOT** = 'IF/ELSE ROOT'

**ITERATION** = 'ITERATION'

**KEYWORD** = 'KEYWORD'

**MESSAGE** = 'MESSAGE'

**RETURN** = 'RETURN'

**SETUP** = 'SETUP'

**TEARDOWN** = 'TEARDOWN'

**TRY** = 'TRY'

**TRY\_EXCEPT\_ROOT** = 'TRY/EXCEPT ROOT'

**WHILE** = 'WHILE'

**config** (*\*\*attributes*)  
Configure model object with given attributes.

`obj.config(name='Example', doc='Something')` is equivalent to setting `obj.name = 'Example'` and `obj.doc = 'Something'`.

New in Robot Framework 4.0.

**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.

**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.

```

    has_setup
    has_teardown
    repr_args = ()
class robot.model.control.TryBranch(type='TRY', patterns=(), pattern_type=None, variable=None, parent=None)
    Bases: robot.model.body.BodyItem
    body_class
        alias of robot.model.body.Body
    repr_args = ('type', 'patterns', 'pattern_type', 'variable')
    type
    patterns
    pattern_type
    variable
    parent
    body
    id
        Branch id omits TRY/EXCEPT root from the parent id part.
    visit(visitor)
    BREAK = 'BREAK'
    CONTINUE = 'CONTINUE'
    ELSE = 'ELSE'
    ELSE_IF = 'ELSE IF'
    EXCEPT = 'EXCEPT'
    FINALLY = 'FINALLY'
    FOR = 'FOR'
    IF = 'IF'
    IF_ELSE_ROOT = 'IF/ELSE ROOT'
    ITERATION = 'ITERATION'
    KEYWORD = 'KEYWORD'
    MESSAGE = 'MESSAGE'
    RETURN = 'RETURN'
    SETUP = 'SETUP'
    TEARDOWN = 'TEARDOWN'
    TRY = 'TRY'
    TRY_EXCEPT_ROOT = 'TRY/EXCEPT ROOT'
    WHILE = 'WHILE'

```

**config** (*\*\*attributes*)

Configure model object with given attributes.

`obj.config(name='Example', doc='Something')` is equivalent to setting `obj.name = 'Example'` and `obj.doc = 'Something'`.

New in Robot Framework 4.0.

**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.

**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.

**has\_setup**

**has\_teardown**

**class** `robot.model.control.Try` (*parent=None*)

Bases: `robot.model.body.BodyItem`

TRY/EXCEPT structure root. Branches are stored in `body`.

**type** = 'TRY/EXCEPT ROOT'

**branch\_class**

alias of `TryBranch`

**branches\_class**

alias of `robot.model.body.Branches`

**parent**

**body**

**try\_branch**

**except\_branches**

**else\_branch**

**finally\_branch**

**id**

Root TRY/EXCEPT id is always None.

**visit** (*visitor*)

**BREAK** = 'BREAK'

**CONTINUE** = 'CONTINUE'

**ELSE** = 'ELSE'

**ELSE\_IF** = 'ELSE IF'

```

EXCEPT = 'EXCEPT'
FINALLY = 'FINALLY'
FOR = 'FOR'
IF = 'IF'
IF_ELSE_ROOT = 'IF/ELSE ROOT'
ITERATION = 'ITERATION'
KEYWORD = 'KEYWORD'
MESSAGE = 'MESSAGE'
RETURN = 'RETURN'
SETUP = 'SETUP'
TEARDOWN = 'TEARDOWN'
TRY = 'TRY'
TRY_EXCEPT_ROOT = 'TRY/EXCEPT ROOT'
WHILE = 'WHILE'

```

**config** (*\*\*attributes*)

Configure model object with given attributes.

`obj.config(name='Example', doc='Something')` is equivalent to setting `obj.name = 'Example'` and `obj.doc = 'Something'`.

New in Robot Framework 4.0.

**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.

**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.

**has\_setup**

**has\_teardown**

**repr\_args** = ()

**class** `robot.model.control.Return` (*values=()*, *parent=None*)

Bases: [robot.model.body.BodyItem](#)

**type** = 'RETURN'

**repr\_args** = ('values',)

**values**

**parent**

**visit** (*visitor*)

**BREAK** = 'BREAK'

**CONTINUE** = 'CONTINUE'

**ELSE** = 'ELSE'

**ELSE\_IF** = 'ELSE IF'

**EXCEPT** = 'EXCEPT'

**FINALLY** = 'FINALLY'

**FOR** = 'FOR'

**IF** = 'IF'

**IF\_ELSE\_ROOT** = 'IF/ELSE ROOT'

**ITERATION** = 'ITERATION'

**KEYWORD** = 'KEYWORD'

**MESSAGE** = 'MESSAGE'

**RETURN** = 'RETURN'

**SETUP** = 'SETUP'

**TEARDOWN** = 'TEARDOWN'

**TRY** = 'TRY'

**TRY\_EXCEPT\_ROOT** = 'TRY/EXCEPT ROOT'

**WHILE** = 'WHILE'

**config** (*\*\*attributes*)

Configure model object with given attributes.

`obj.config(name='Example', doc='Something')` is equivalent to setting `obj.name = 'Example'` and `obj.doc = 'Something'`.

New in Robot Framework 4.0.

**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.

**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.

**has\_setup**

**has\_teardown**



**id**

Item id in format like s1-t3-k1.

See [TestSuite.id](#) for more information.

**class** `robot.model.control.Continue` (*parent=None*)

Bases: `robot.model.body.BodyItem`

**type** = 'CONTINUE'

**parent**

**BREAK** = 'BREAK'

**CONTINUE** = 'CONTINUE'

**ELSE** = 'ELSE'

**ELSE\_IF** = 'ELSE IF'

**EXCEPT** = 'EXCEPT'

**FINALLY** = 'FINALLY'

**FOR** = 'FOR'

**IF** = 'IF'

**IF\_ELSE\_ROOT** = 'IF/ELSE ROOT'

**ITERATION** = 'ITERATION'

**KEYWORD** = 'KEYWORD'

**MESSAGE** = 'MESSAGE'

**RETURN** = 'RETURN'

**SETUP** = 'SETUP'

**TEARDOWN** = 'TEARDOWN'

**TRY** = 'TRY'

**TRY\_EXCEPT\_ROOT** = 'TRY/EXCEPT ROOT'

**WHILE** = 'WHILE'

**config** (*\*\*attributes*)

Configure model object with given attributes.

`obj.config(name='Example', doc='Something')` is equivalent to setting `obj.name = 'Example'` and `obj.doc = 'Something'`.

New in Robot Framework 4.0.

**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.

**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.

**has\_setup**

**has\_teardown**

**id**

Item id in format like s1-t3-k1.

See `TestSuite.id` for more information.

**repr\_args** = ()

**visit** (*visitor*)

**class** `robot.model.control.Break` (*parent=None*)

Bases: `robot.model.body.BodyItem`

**BREAK** = 'BREAK'

**CONTINUE** = 'CONTINUE'

**ELSE** = 'ELSE'

**ELSE\_IF** = 'ELSE IF'

**EXCEPT** = 'EXCEPT'

**FINALLY** = 'FINALLY'

**FOR** = 'FOR'

**IF** = 'IF'

**IF\_ELSE\_ROOT** = 'IF/ELSE ROOT'

**ITERATION** = 'ITERATION'

**KEYWORD** = 'KEYWORD'

**MESSAGE** = 'MESSAGE'

**RETURN** = 'RETURN'

**SETUP** = 'SETUP'

**TEARDOWN** = 'TEARDOWN'

**TRY** = 'TRY'

**TRY\_EXCEPT\_ROOT** = 'TRY/EXCEPT ROOT'

**WHILE** = 'WHILE'

**config** (*\*\*attributes*)

Configure model object with given attributes.

`obj.config(name='Example', doc='Something')` is equivalent to setting `obj.name = 'Example'` and `obj.doc = 'Something'`.

New in Robot Framework 4.0.

**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.

**`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.

**`has_setup`**

**`has_teardown`**

**`id`**

Item id in format like s1-t3-k1.

See `TestSuite.id` for more information.

**`repr_args = ()`**

**`type = 'BREAK'`**

**`parent`**

**`visit(visitor)`**

## robot.model.filter module

**class** `robot.model.filter.EmptySuiteRemover` (*preserve\_direct\_children=False*)

Bases: `robot.model.visitor.SuiteVisitor`

**`end_suite(suite)`**

Called when a suite ends. Default implementation does nothing.

**`visit_test(test)`**

Implements traversing through tests.

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

**`visit_keyword(kw)`**

Implements traversing through keywords.

Can be overridden to allow modifying the passed in `kw` without calling `start_keyword()` or `end_keyword()` nor visiting the body of the keyword

**`end_body_item(item)`**

Called, by default, when keywords, messages or control structures end.

More specific `end_keyword()`, `end_message()`, `:meth:'end_for'`, etc. can be implemented to visit only keywords, messages or specific control structures.

Default implementation does nothing.

**`end_break(break_)`**

Called when a BREAK element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_continue** (*continue\_*)  
Called when a CONTINUE element ends.  
By default, calls `end_body_item()` which, by default, does nothing.

**end\_for** (*for\_*)  
Called when a FOR loop ends.  
By default, calls `end_body_item()` which, by default, does nothing.

**end\_for\_iteration** (*iteration*)  
Called when a FOR loop iteration ends.  
By default, calls `end_body_item()` which, by default, does nothing.

**end\_if** (*if\_*)  
Called when an IF/ELSE structure ends.  
By default, calls `end_body_item()` which, by default, does nothing.

**end\_if\_branch** (*branch*)  
Called when an IF/ELSE branch ends.  
By default, calls `end_body_item()` which, by default, does nothing.

**end\_keyword** (*keyword*)  
Called when a keyword ends.  
By default, calls `end_body_item()` which, by default, does nothing.

**end\_message** (*msg*)  
Called when a message ends.  
By default, calls `end_body_item()` which, by default, does nothing.

**end\_return** (*return\_*)  
Called when a RETURN element ends.  
By default, calls `end_body_item()` which, by default, does nothing.

**end\_test** (*test*)  
Called when a test ends. Default implementation does nothing.

**end\_try** (*try\_*)  
Called when a TRY/EXCEPT structure ends.  
By default, calls `end_body_item()` which, by default, does nothing.

**end\_try\_branch** (*branch*)  
Called when TRY, EXCEPT, ELSE and FINALLY branches end.  
By default, calls `end_body_item()` which, by default, does nothing.

**end\_while** (*while\_*)  
Called when a WHILE loop ends.  
By default, calls `end_body_item()` which, by default, does nothing.

**end\_while\_iteration** (*iteration*)  
Called when a WHILE loop iteration ends.  
By default, calls `end_body_item()` which, by default, does nothing.

**start\_body\_item** (*item*)  
Called, by default, when keywords, messages or control structures start.

More specific `start_keyword()`, `start_message()`, `:meth:'start_for`, etc. can be implemented to visit only keywords, messages or specific control structures.

Can return explicit `False` to stop visiting. Default implementation does nothing.

**start\_break** (*break\_*)

Called when a BREAK element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_continue** (*continue\_*)

Called when a CONTINUE element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_for** (*for\_*)

Called when a FOR loop starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_for\_iteration** (*iteration*)

Called when a FOR loop iteration starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_if** (*if\_*)

Called when an IF/ELSE structure starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_if\_branch** (*branch*)

Called when an IF/ELSE branch starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_keyword** (*keyword*)

Called when a keyword starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_message** (*msg*)

Called when a message starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_return** (*return\_*)

Called when a RETURN element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_suite** (*suite*)

Called when a suite starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

**start\_test** (*test*)

Called when a test starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

**start\_try** (*try\_*)

Called when a TRY/EXCEPT structure starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_try\_branch** (*branch*)

Called when TRY, EXCEPT, ELSE or FINALLY branches start.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_while** (*while\_*)

Called when a WHILE loop starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_while\_iteration** (*iteration*)

Called when a WHILE loop iteration starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**visit\_break** (*break\_*)

Visits BREAK elements.

**visit\_continue** (*continue\_*)

Visits CONTINUE elements.

**visit\_for** (*for\_*)

Implements traversing through FOR loops.

Can be overridden to allow modifying the passed in `for_` without calling `start_for()` or `end_for()` nor visiting body.

**visit\_for\_iteration** (*iteration*)

Implements traversing through single FOR loop iteration.

This is only used with the result side model because on the running side there are no iterations.

Can be overridden to allow modifying the passed in `iteration` without calling `start_for_iteration()` or `end_for_iteration()` nor visiting body.

**visit\_if** (*if\_*)

Implements traversing through IF/ELSE structures.

Notice that `if_` does not have any data directly. Actual IF/ELSE branches are in its body and visited using `visit_if_branch()`.

Can be overridden to allow modifying the passed in `if_` without calling `start_if()` or `end_if()` nor visiting branches.

**visit\_if\_branch** (*branch*)

Implements traversing through single IF/ELSE branch.

Can be overridden to allow modifying the passed in *branch* without calling `start_if_branch()` or `end_if_branch()` nor visiting body.

**visit\_message** (*msg*)

Implements visiting messages.

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

**visit\_return** (*return\_*)

Visits a RETURN elements.

**visit\_suite** (*suite*)

Implements traversing through suites.

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

**visit\_try** (*try\_*)

Implements traversing through TRY/EXCEPT structures.

This method is used with the TRY/EXCEPT root element. Actual TRY, EXCEPT, ELSE and FINALLY branches are visited separately using `visit_try_branch()`.

**visit\_try\_branch** (*branch*)

Visits individual TRY, EXCEPT, ELSE and FINALLY branches.

**visit\_while** (*while\_*)

Implements traversing through WHILE loops.

Can be overridden to allow modifying the passed in *while\_* without calling `start_while()` or `end_while()` nor visiting body.

**visit\_while\_iteration** (*iteration*)

Implements traversing through single WHILE loop iteration.

This is only used with the result side model because on the running side there are no iterations.

Can be overridden to allow modifying the passed in *iteration* without calling `start_while_iteration()` or `end_while_iteration()` nor visiting body.

```
class robot.model.filter.Filter (include_suites=None,      include_tests=None,      in-
                                clude_tags=None, exclude_tags=None)
```

Bases: `robot.model.filter.EmptySuiteRemover`

**include\_suites**

**include\_tests**

**include\_tags**

**exclude\_tags**

**start\_suite** (*suite*)

Called when a suite starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

**end\_body\_item** (*item*)

Called, by default, when keywords, messages or control structures end.

More specific `end_keyword()`, `end_message()`, `:meth:'end_for'`, etc. can be implemented to visit only keywords, messages or specific control structures.

Default implementation does nothing.

**end\_break** (*break\_*)

Called when a BREAK element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_continue** (*continue\_*)

Called when a CONTINUE element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_for** (*for\_*)

Called when a FOR loop ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_for\_iteration** (*iteration*)

Called when a FOR loop iteration ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_if** (*if\_*)

Called when an IF/ELSE structure ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_if\_branch** (*branch*)

Called when an IF/ELSE branch ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_keyword** (*keyword*)

Called when a keyword ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_message** (*msg*)

Called when a message ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_return** (*return\_*)

Called when a RETURN element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_suite** (*suite*)

Called when a suite ends. Default implementation does nothing.

**end\_test** (*test*)

Called when a test ends. Default implementation does nothing.

**end\_try** (*try\_*)

Called when a TRY/EXCEPT structure ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_try\_branch** (*branch*)

Called when TRY, EXCEPT, ELSE and FINALLY branches end.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_while** (*while\_*)

Called when a WHILE loop ends.

By default, calls `end_body_item()` which, by default, does nothing.



**end\_while\_iteration** (*iteration*)

Called when a WHILE loop iteration ends.

By default, calls `end_body_item()` which, by default, does nothing.

**start\_body\_item** (*item*)

Called, by default, when keywords, messages or control structures start.

More specific `start_keyword()`, `start_message()`, `:meth:'start_for`, etc. can be implemented to visit only keywords, messages or specific control structures.

Can return explicit `False` to stop visiting. Default implementation does nothing.

**start\_break** (*break\_*)

Called when a BREAK element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_continue** (*continue\_*)

Called when a CONTINUE element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_for** (*for\_*)

Called when a FOR loop starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_for\_iteration** (*iteration*)

Called when a FOR loop iteration starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_if** (*if\_*)

Called when an IF/ELSE structure starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_if\_branch** (*branch*)

Called when an IF/ELSE branch starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_keyword** (*keyword*)

Called when a keyword starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_message** (*msg*)

Called when a message starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_return** (*return\_*)

Called when a RETURN element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_test** (*test*)

Called when a test starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

**start\_try** (*try\_*)

Called when a TRY/EXCEPT structure starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_try\_branch** (*branch*)

Called when TRY, EXCEPT, ELSE or FINALLY branches start.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_while** (*while\_*)

Called when a WHILE loop starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_while\_iteration** (*iteration*)

Called when a WHILE loop iteration starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**visit\_break** (*break\_*)

Visits BREAK elements.

**visit\_continue** (*continue\_*)

Visits CONTINUE elements.

**visit\_for** (*for\_*)

Implements traversing through FOR loops.

Can be overridden to allow modifying the passed in `for_` without calling `start_for()` or `end_for()` nor visiting body.

**visit\_for\_iteration** (*iteration*)

Implements traversing through single FOR loop iteration.

This is only used with the result side model because on the running side there are no iterations.

Can be overridden to allow modifying the passed in `iteration` without calling `start_for_iteration()` or `end_for_iteration()` nor visiting body.

**visit\_if** (*if\_*)

Implements traversing through IF/ELSE structures.

Notice that `if_` does not have any data directly. Actual IF/ELSE branches are in its `body` and visited using `visit_if_branch()`.

Can be overridden to allow modifying the passed in `if_` without calling `start_if()` or `end_if()` nor visiting branches.

**visit\_if\_branch** (*branch*)

Implements traversing through single IF/ELSE branch.

Can be overridden to allow modifying the passed in `branch` without calling `start_if_branch()` or `end_if_branch()` nor visiting body.

**visit\_keyword** (*kw*)

Implements traversing through keywords.

Can be overridden to allow modifying the passed in `kw` without calling `start_keyword()` or `end_keyword()` nor visiting the body of the keyword

**visit\_message** (*msg*)

Implements visiting messages.

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

**visit\_return** (*return\_*)

Visits a RETURN elements.

**visit\_suite** (*suite*)

Implements traversing through suites.

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

**visit\_test** (*test*)

Implements traversing through tests.

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

**visit\_try** (*try\_*)

Implements traversing through TRY/EXCEPT structures.

This method is used with the TRY/EXCEPT root element. Actual TRY, EXCEPT, ELSE and FINALLY branches are visited separately using `visit_try_branch()`.

**visit\_try\_branch** (*branch*)

Visits individual TRY, EXCEPT, ELSE and FINALLY branches.

**visit\_while** (*while\_*)

Implements traversing through WHILE loops.

Can be overridden to allow modifying the passed in `while_` without calling `start_while()` or `end_while()` nor visiting body.

**visit\_while\_iteration** (*iteration*)

Implements traversing through single WHILE loop iteration.

This is only used with the result side model because on the running side there are no iterations.

Can be overridden to allow modifying the passed in `iteration` without calling `start_while_iteration()` or `end_while_iteration()` nor visiting body.

## robot.model.fixture module

`robot.model.fixture.create_fixture(fixture, parent, type)`

**robot.model.itemlist module**

```
class robot.model.itemlist.ItemList (item_class, common_attrs=None, items=None)
    Bases: collections.abc.MutableSequence

    create (*args, **kwargs)

    append (item)
        S.append(value) – append value to the end of the sequence

    extend (items)
        S.extend(iterable) – extend sequence by appending elements from the iterable

    insert (index, item)
        S.insert(index, value) – insert value before index

    index (value [, start [, stop ] ] ) → integer – return first index of value.
        Raises ValueError if the value is not present.

        Supporting start and stop arguments is optional, but recommended.

    clear () → None – remove all items from S

    visit (visitor)

    count (value) → integer – return number of occurrences of value

    sort ()

    reverse ()
        S.reverse() – reverse IN PLACE

    pop ([index ]) → item – remove and return item at index (default last).
        Raise IndexError if list is empty or index is out of range.

    remove (value)
        S.remove(value) – remove first occurrence of value. Raise ValueError if the value is not present.
```

**robot.model.keyword module**

```
class robot.model.keyword.Keyword (name="", doc="", args=(), assign=(), tags=(), time-
                                   out=None, type='KEYWORD', parent=None)
    Bases: robot.model.body.BodyItem

    Base model for a single keyword.

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

    repr_args = ('name', 'args', 'assign')

    doc

    args

    assign

    timeout

    type

    parent

    name
```

**teardown**

Keyword teardown as a *Keyword* object.

Teardown can be modified by setting attributes directly:

```
keyword.teardown.name = 'Example'
keyword.teardown.args = ('First', 'Second')
```

Alternatively the *config()* method can be used to set multiple attributes in one call:

```
keyword.teardown.config(name='Example', args=('First', 'Second'))
```

The easiest way to reset the whole teardown is setting it to *None*. It will automatically recreate the underlying *Keyword* object:

```
keyword.teardown = None
```

This attribute is a *Keyword* object also when a keyword has no teardown but in that case its truth value is *False*. If there is a need to just check does a keyword have a teardown, using the *has\_teardown* attribute avoids creating the *Keyword* object and is thus more memory efficient.

New in Robot Framework 4.0. Earlier teardown was accessed like `keyword.keywords.teardown`. *has\_teardown* is new in Robot Framework 4.1.2.

**has\_teardown**

Check does a keyword have a teardown without creating a teardown object.

A difference between using `if kw.has_teardown:` and `if kw.teardown:` is that accessing the *teardown* attribute creates a *Keyword* object representing a teardown even when the keyword actually does not have one. This typically does not matter, but with bigger suite structures having lot of keywords it can have a considerable effect on memory usage.

New in Robot Framework 4.1.2.

**tags**

Keyword tags as a *Tags* object.

**visit** (*visitor*)

*Visitor interface* entry-point.

**BREAK** = 'BREAK'

**CONTINUE** = 'CONTINUE'

**ELSE** = 'ELSE'

**ELSE\_IF** = 'ELSE IF'

**EXCEPT** = 'EXCEPT'

**FINALLY** = 'FINALLY'

**FOR** = 'FOR'

**IF** = 'IF'

**IF\_ELSE\_ROOT** = 'IF/ELSE ROOT'

**ITERATION** = 'ITERATION'

**KEYWORD** = 'KEYWORD'

**MESSAGE** = 'MESSAGE'

**RETURN** = 'RETURN'

**SETUP** = 'SETUP'

**TEARDOWN** = 'TEARDOWN'

**TRY** = 'TRY'

**TRY\_EXCEPT\_ROOT** = 'TRY/EXCEPT ROOT'

**WHILE** = 'WHILE'

**config** (\*\*attributes)

Configure model object with given attributes.

`obj.config(name='Example', doc='Something')` is equivalent to setting `obj.name = 'Example'` and `obj.doc = 'Something'`.

New in Robot Framework 4.0.

**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.

**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.

**has\_setup**

**id**

Item id in format like `s1-t3-k1`.

See [`TestSuite.id`](#) for more information.

**class** `robot.model.keyword.Keywords` (*parent=None, keywords=None*)

Bases: [`robot.model.itemlist.ItemList`](#)

A list-like object representing keywords in a suite, a test or a keyword.

Read-only and deprecated since Robot Framework 4.0.

**deprecation\_message** = "'keywords' attribute is read-only and deprecated since Robot Framework 4.0"

**setup**

**create\_setup** (\*args, \*\*kwargs)

**teardown**

**create\_teardown** (\*args, \*\*kwargs)

**all**

Iterates over all keywords, including setup and teardown.

**normal**

Iterates over normal keywords, omitting setup and teardown.

**create** (\*args, \*\*kwargs)

**append** (*item*)  
 S.append(value) – append value to the end of the sequence

**extend** (*items*)  
 S.extend(iterable) – extend sequence by appending elements from the iterable

**insert** (*index*, *item*)  
 S.insert(index, value) – insert value before index

**pop** ([*index*]) → *item* – remove and return item at index (default last).  
 Raise IndexError if list is empty or index is out of range.

**remove** (*item*)  
 S.remove(value) – remove first occurrence of value. Raise ValueError if the value is not present.

**clear** () → None – remove all items from S

**count** (*value*) → integer – return number of occurrences of value

**index** (*value*[, *start*[, *stop*]]) → integer – return first index of value.  
 Raises ValueError if the value is not present.  
 Supporting start and stop arguments is optional, but recommended.

**visit** (*visitor*)

**sort** ()

**reverse** ()  
 S.reverse() – reverse *IN PLACE*

**classmethod raise\_deprecation\_error** ()

### robot.model.message module

**class** robot.model.message.**Message** (*message*=", *level*='INFO', *html*=False, *timestamp*=None, *parent*=None)  
 Bases: *robot.model.body.BodyItem*

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.

**type** = 'MESSAGE'

**repr\_args** = ('message', 'level')

**message**  
 The message content as a string.

**level**  
 Severity of the message. Either TRACE, DEBUG, INFO, WARN, ERROR, FAIL or “SKIP”. The last two are 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.

**id**

Item id in format like s1-t3-k1.

See *TestSuite.id* for more information.

**visit** (*visitor*)

*Visitor interface* entry-point.

**BREAK** = 'BREAK'

**CONTINUE** = 'CONTINUE'

**ELSE** = 'ELSE'

**ELSE\_IF** = 'ELSE IF'

**EXCEPT** = 'EXCEPT'

**FINALLY** = 'FINALLY'

**FOR** = 'FOR'

**IF** = 'IF'

**IF\_ELSE\_ROOT** = 'IF/ELSE ROOT'

**ITERATION** = 'ITERATION'

**KEYWORD** = 'KEYWORD'

**MESSAGE** = 'MESSAGE'

**RETURN** = 'RETURN'

**SETUP** = 'SETUP'

**TEARDOWN** = 'TEARDOWN'

**TRY** = 'TRY'

**TRY\_EXCEPT\_ROOT** = 'TRY/EXCEPT ROOT'

**WHILE** = 'WHILE'

**config** (*\*\*attributes*)

Configure model object with given attributes.

`obj.config(name='Example', doc='Something')` is equivalent to setting `obj.name = 'Example'` and `obj.doc = 'Something'`.

New in Robot Framework 4.0.

**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.

**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.

**has\_setup**

**has\_teardown**

**class** `robot.model.message.Messages` (`message_class=<class 'robot.model.message.Message'>`,  
`parent=None, messages=None`)

Bases: `robot.model.itemlist.ItemList`

**append** (`item`)

S.append(value) – append value to the end of the sequence

**clear** () → None – remove all items from S

**count** (`value`) → integer – return number of occurrences of value

**create** (`*args, **kwargs`)

**extend** (`items`)

S.extend(iterable) – extend sequence by appending elements from the iterable

**index** (`value` [, `start` [, `stop` ]]) → integer – return first index of value.

Raises `ValueError` if the value is not present.

Supporting start and stop arguments is optional, but recommended.

**insert** (`index, item`)

S.insert(index, value) – insert value before index

**pop** ([`index`]) → item – remove and return item at index (default last).

Raise `IndexError` if list is empty or index is out of range.

**remove** (`value`)

S.remove(value) – remove first occurrence of value. Raise `ValueError` if the value is not present.

**reverse** ()

S.reverse() – reverse *IN PLACE*

**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** () → a set-like object providing a view on D's items

**keys** () → a set-like object providing a view on 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**() → an object providing a view on `D`'s values

## robot.model.modelobject module

**class** `robot.model.modelobject.ModelObject`

Bases: `object`

**repr\_args** = ()

**config**(*\*\*attributes*)

Configure model object with given attributes.

`obj.config(name='Example', doc='Something')` is equivalent to setting `obj.name = 'Example'` and `obj.doc = 'Something'`.

New in Robot Framework 4.0.

**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.

**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.

`robot.model.modelobject.full_name(obj)`

## robot.model.modifier module

**class** `robot.model.modifier.ModelModifier`(*visitors*, *empty\_suite\_ok*, *logger*)

Bases: `robot.model.visitor.SuiteVisitor`

**visit\_suite**(*suite*)

Implements traversing through suites.

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

**end\_body\_item**(*item*)

Called, by default, when keywords, messages or control structures end.

More specific `end_keyword()`, `end_message()`, `:meth:'end_for`, etc. can be implemented to visit only keywords, messages or specific control structures.

Default implementation does nothing.

**end\_break** (*break\_*)

Called when a BREAK element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_continue** (*continue\_*)

Called when a CONTINUE element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_for** (*for\_*)

Called when a FOR loop ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_for\_iteration** (*iteration*)

Called when a FOR loop iteration ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_if** (*if\_*)

Called when an IF/ELSE structure ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_if\_branch** (*branch*)

Called when an IF/ELSE branch ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_keyword** (*keyword*)

Called when a keyword ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_message** (*msg*)

Called when a message ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_return** (*return\_*)

Called when a RETURN element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_suite** (*suite*)

Called when a suite ends. Default implementation does nothing.

**end\_test** (*test*)

Called when a test ends. Default implementation does nothing.

**end\_try** (*try\_*)

Called when a TRY/EXCEPT structure ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_try\_branch** (*branch*)

Called when TRY, EXCEPT, ELSE and FINALLY branches end.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_while** (*while\_*)

Called when a WHILE loop ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_while\_iteration** (*iteration*)

Called when a WHILE loop iteration ends.

By default, calls `end_body_item()` which, by default, does nothing.

**start\_body\_item** (*item*)

Called, by default, when keywords, messages or control structures start.

More specific `start_keyword()`, `start_message()`, `:meth:'start_for`, etc. can be implemented to visit only keywords, messages or specific control structures.

Can return explicit `False` to stop visiting. Default implementation does nothing.

**start\_break** (*break\_*)

Called when a BREAK element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_continue** (*continue\_*)

Called when a CONTINUE element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_for** (*for\_*)

Called when a FOR loop starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_for\_iteration** (*iteration*)

Called when a FOR loop iteration starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_if** (*if\_*)

Called when an IF/ELSE structure starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_if\_branch** (*branch*)

Called when an IF/ELSE branch starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_keyword** (*keyword*)

Called when a keyword starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_message** (*msg*)

Called when a message starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_return** (*return\_*)

Called when a RETURN element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_suite** (*suite*)

Called when a suite starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

**start\_test** (*test*)

Called when a test starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

**start\_try** (*try\_*)

Called when a TRY/EXCEPT structure starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_try\_branch** (*branch*)

Called when TRY, EXCEPT, ELSE or FINALLY branches start.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_while** (*while\_*)

Called when a WHILE loop starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_while\_iteration** (*iteration*)

Called when a WHILE loop iteration starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**visit\_break** (*break\_*)

Visits BREAK elements.

**visit\_continue** (*continue\_*)

Visits CONTINUE elements.

**visit\_for** (*for\_*)

Implements traversing through FOR loops.

Can be overridden to allow modifying the passed in `for_` without calling `start_for()` or `end_for()` nor visiting body.

**visit\_for\_iteration** (*iteration*)

Implements traversing through single FOR loop iteration.

This is only used with the result side model because on the running side there are no iterations.

Can be overridden to allow modifying the passed in `iteration` without calling `start_for_iteration()` or `end_for_iteration()` nor visiting body.

**visit\_if** (*if\_*)

Implements traversing through IF/ELSE structures.

Notice that `if_` does not have any data directly. Actual IF/ELSE branches are in its body and visited using `visit_if_branch()`.

Can be overridden to allow modifying the passed in `if_` without calling `start_if()` or `end_if()` nor visiting branches.

**visit\_if\_branch** (*branch*)

Implements traversing through single IF/ELSE branch.

Can be overridden to allow modifying the passed in `branch` without calling `start_if_branch()` or `end_if_branch()` nor visiting body.

**visit\_keyword** (*kw*)

Implements traversing through keywords.

Can be overridden to allow modifying the passed in `kw` without calling `start_keyword()` or `end_keyword()` nor visiting the body of the keyword

**visit\_message** (*msg*)

Implements visiting messages.

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

**visit\_return** (*return\_*)

Visits a RETURN elements.

**visit\_test** (*test*)

Implements traversing through tests.

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

**visit\_try** (*try\_*)

Implements traversing through TRY/EXCEPT structures.

This method is used with the TRY/EXCEPT root element. Actual TRY, EXCEPT, ELSE and FINALLY branches are visited separately using `visit_try_branch()`.

**visit\_try\_branch** (*branch*)

Visits individual TRY, EXCEPT, ELSE and FINALLY branches.

**visit\_while** (*while\_*)

Implements traversing through WHILE loops.

Can be overridden to allow modifying the passed in `while_` without calling `start_while()` or `end_while()` nor visiting body.

**visit\_while\_iteration** (*iteration*)

Implements traversing through single WHILE loop iteration.

This is only used with the result side model because on the running side there are no iterations.

Can be overridden to allow modifying the passed in `iteration` without calling `start_while_iteration()` or `end_while_iteration()` nor visiting body.

**robot.model.namepatterns module**

```

class robot.model.namepatterns.NamePatterns (patterns=None)
    Bases: object

    match (name, longname=None)

class robot.model.namepatterns.SuiteNamePatterns (patterns=None)
    Bases: robot.model.namepatterns.NamePatterns

    match (name, longname=None)

class robot.model.namepatterns.TestNamePatterns (patterns=None)
    Bases: robot.model.namepatterns.NamePatterns

    match (name, longname=None)

```

**robot.model.statistics module**

```

class robot.model.statistics.Statistics (suite, suite_stat_level=-1, tag_stat_include=None,
                                         tag_stat_exclude=None, tag_stat_combine=None,
                                         tag_doc=None, tag_stat_link=None, rpa=False)

    Bases: object

    Container for total, suite and tag statistics.

    Accepted parameters have the same semantics as the matching command line options.

    total = None
        Instance of TotalStatistics.

    suite = None
        Instance of SuiteStatistics.

    tags = None
        Instance of TagStatistics.

    visit (visitor)

class robot.model.statistics.StatisticsBuilder (total_builder, suite_builder,
                                                tag_builder)

    Bases: robot.model.visitor.SuiteVisitor

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

        Can return explicit False to stop visiting.

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

    visit_test (test)
        Implements traversing through tests.

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

    visit_keyword (kw)
        Implements traversing through keywords.

        Can be overridden to allow modifying the passed in kw without calling start_keyword() or
        end_keyword() nor visiting the body of the keyword

```

**end\_body\_item** (*item*)

Called, by default, when keywords, messages or control structures end.

More specific `end_keyword()`, `end_message()`, `:meth:'end_for`, etc. can be implemented to visit only keywords, messages or specific control structures.

Default implementation does nothing.

**end\_break** (*break\_*)

Called when a BREAK element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_continue** (*continue\_*)

Called when a CONTINUE element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_for** (*for\_*)

Called when a FOR loop ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_for\_iteration** (*iteration*)

Called when a FOR loop iteration ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_if** (*if\_*)

Called when an IF/ELSE structure ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_if\_branch** (*branch*)

Called when an IF/ELSE branch ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_keyword** (*keyword*)

Called when a keyword ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_message** (*msg*)

Called when a message ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_return** (*return\_*)

Called when a RETURN element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_test** (*test*)

Called when a test ends. Default implementation does nothing.

**end\_try** (*try\_*)

Called when a TRY/EXCEPT structure ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_try\_branch** (*branch*)

Called when TRY, EXCEPT, ELSE and FINALLY branches end.

By default, calls `end_body_item()` which, by default, does nothing.



**end\_while** (*while\_*)

Called when a WHILE loop ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_while\_iteration** (*iteration*)

Called when a WHILE loop iteration ends.

By default, calls `end_body_item()` which, by default, does nothing.

**start\_body\_item** (*item*)

Called, by default, when keywords, messages or control structures start.

More specific `start_keyword()`, `start_message()`, `:meth:'start_for`, etc. can be implemented to visit only keywords, messages or specific control structures.

Can return explicit `False` to stop visiting. Default implementation does nothing.

**start\_break** (*break\_*)

Called when a BREAK element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_continue** (*continue\_*)

Called when a CONTINUE element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_for** (*for\_*)

Called when a FOR loop starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_for\_iteration** (*iteration*)

Called when a FOR loop iteration starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_if** (*if\_*)

Called when an IF/ELSE structure starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_if\_branch** (*branch*)

Called when an IF/ELSE branch starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_keyword** (*keyword*)

Called when a keyword starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_message** (*msg*)

Called when a message starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_return** (*return\_*)

Called when a RETURN element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_test** (*test*)

Called when a test starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

**start\_try** (*try\_*)

Called when a TRY/EXCEPT structure starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_try\_branch** (*branch*)

Called when TRY, EXCEPT, ELSE or FINALLY branches start.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_while** (*while\_*)

Called when a WHILE loop starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_while\_iteration** (*iteration*)

Called when a WHILE loop iteration starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**visit\_break** (*break\_*)

Visits BREAK elements.

**visit\_continue** (*continue\_*)

Visits CONTINUE elements.

**visit\_for** (*for\_*)

Implements traversing through FOR loops.

Can be overridden to allow modifying the passed in `for_` without calling `start_for()` or `end_for()` nor visiting body.

**visit\_for\_iteration** (*iteration*)

Implements traversing through single FOR loop iteration.

This is only used with the result side model because on the running side there are no iterations.

Can be overridden to allow modifying the passed in `iteration` without calling `start_for_iteration()` or `end_for_iteration()` nor visiting body.

**visit\_if**(*if\_*)

Implements traversing through IF/ELSE structures.

Notice that *if\_* does not have any data directly. Actual IF/ELSE branches are in its body and visited using *visit\_if\_branch()*.

Can be overridden to allow modifying the passed in *if\_* without calling *start\_if()* or *end\_if()* nor visiting branches.

**visit\_if\_branch**(*branch*)

Implements traversing through single IF/ELSE branch.

Can be overridden to allow modifying the passed in *branch* without calling *start\_if\_branch()* or *end\_if\_branch()* nor visiting body.

**visit\_message**(*msg*)

Implements visiting messages.

Can be overridden to allow modifying the passed in *msg* without calling *start\_message()* or *end\_message()*.

**visit\_return**(*return\_*)

Visits a RETURN elements.

**visit\_suite**(*suite*)

Implements traversing through suites.

Can be overridden to allow modifying the passed in *suite* without calling *start\_suite()* or *end\_suite()* nor visiting child suites, tests or setup and teardown at all.

**visit\_try**(*try\_*)

Implements traversing through TRY/EXCEPT structures.

This method is used with the TRY/EXCEPT root element. Actual TRY, EXCEPT, ELSE and FINALLY branches are visited separately using *visit\_try\_branch()*.

**visit\_try\_branch**(*branch*)

Visits individual TRY, EXCEPT, ELSE and FINALLY branches.

**visit\_while**(*while\_*)

Implements traversing through WHILE loops.

Can be overridden to allow modifying the passed in *while\_* without calling *start\_while()* or *end\_while()* nor visiting body.

**visit\_while\_iteration**(*iteration*)

Implements traversing through single WHILE loop iteration.

This is only used with the result side model because on the running side there are no iterations.

Can be overridden to allow modifying the passed in *iteration* without calling *start\_while\_iteration()* or *end\_while\_iteration()* nor visiting body.

**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. All 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.

skipped = None
    Number of skipped tests.

elapsed = None
    Number of milliseconds it took to execute.

get_attributes (include_label=False, include_elapsed=False, exclude_empty=True, val-
                  ues_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, val-
                    ues_as_strings=False, html_escape=False)

    total

    visit (visitor)

class robot.model.stats.SuiteStat (suite)
    Bases: robot.model.stats.Stat

    Stores statistics values for a single suite.

    type = 'suite'

    id = None
        Identifier of the suite, e.g. s1-s2.

    elapsed = None
        Number of milliseconds it took to execute this suite, including sub-suites.

    add_stat (other)

    add_test (test)

    get_attributes (include_label=False, include_elapsed=False, exclude_empty=True, val-
                    ues_as_strings=False, html_escape=False)

    total

    visit (visitor)

class robot.model.stats.TagStat (name, doc="", links=None, 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.

**combined** = None

Pattern as a string if the tag is combined, None otherwise.

**info**

Returns additional information of the tag statistics are about. Either *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 *combined* or an empty string.

**total**

**type** = 'tag'

**visit** (*visitor*)

### robot.model.sitestatistics module

**class** robot.model.sitestatistics.**SuiteStatistics** (*suite*)

Bases: object

Container for suite statistics.

**stat** = None

Instance of [SuiteStat](#).

**suites** = None

List of [TestSuite](#) objects.

**visit** (*visitor*)

**class** robot.model.sitestatistics.**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 a suite starts. Default implementation does nothing.
        Can return explicit False to stop visiting.
    visit_test(test)
        Implements traversing through tests.
        Can be overridden to allow modifying the passed in test without calling start_test() or
        end_test() nor visiting the body of the test.
```

**visit\_keyword** (*keyword*)

Implements traversing through keywords.

Can be overridden to allow modifying the passed in *kw* without calling `start_keyword()` or `end_keyword()` nor visiting the body of the keyword

**end\_body\_item** (*item*)

Called, by default, when keywords, messages or control structures end.

More specific `end_keyword()`, `end_message()`, `:meth:'end_for`, etc. can be implemented to visit only keywords, messages or specific control structures.

Default implementation does nothing.

**end\_break** (*break\_*)

Called when a BREAK element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_continue** (*continue\_*)

Called when a CONTINUE element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_for** (*for\_*)

Called when a FOR loop ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_for\_iteration** (*iteration*)

Called when a FOR loop iteration ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_if** (*if\_*)

Called when an IF/ELSE structure ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_if\_branch** (*branch*)

Called when an IF/ELSE branch ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_keyword** (*keyword*)

Called when a keyword ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_message** (*msg*)

Called when a message ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_return** (*return\_*)

Called when a RETURN element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_suite** (*suite*)

Called when a suite ends. Default implementation does nothing.

**end\_test** (*test*)

Called when a test ends. Default implementation does nothing.

**end\_try** (*try\_*)

Called when a TRY/EXCEPT structure ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_try\_branch** (*branch*)

Called when TRY, EXCEPT, ELSE and FINALLY branches end.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_while** (*while\_*)

Called when a WHILE loop ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_while\_iteration** (*iteration*)

Called when a WHILE loop iteration ends.

By default, calls `end_body_item()` which, by default, does nothing.

**start\_body\_item** (*item*)

Called, by default, when keywords, messages or control structures start.

More specific `start_keyword()`, `start_message()`, `:meth:'start_for`, etc. can be implemented to visit only keywords, messages or specific control structures.

Can return explicit `False` to stop visiting. Default implementation does nothing.

**start\_break** (*break\_*)

Called when a BREAK element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_continue** (*continue\_*)

Called when a CONTINUE element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_for** (*for\_*)

Called when a FOR loop starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_for\_iteration** (*iteration*)

Called when a FOR loop iteration starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_if** (*if\_*)

Called when an IF/ELSE structure starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_if\_branch** (*branch*)

Called when an IF/ELSE branch starts.

By default, calls `start_body_item()` which, by default, does nothing.



Can return explicit `False` to stop visiting.

**start\_keyword** (*keyword*)

Called when a keyword starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_message** (*msg*)

Called when a message starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_return** (*return\_*)

Called when a RETURN element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_test** (*test*)

Called when a test starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

**start\_try** (*try\_*)

Called when a TRY/EXCEPT structure starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_try\_branch** (*branch*)

Called when TRY, EXCEPT, ELSE or FINALLY branches start.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_while** (*while\_*)

Called when a WHILE loop starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_while\_iteration** (*iteration*)

Called when a WHILE loop iteration starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**visit\_break** (*break\_*)

Visits BREAK elements.

**visit\_continue** (*continue\_*)

Visits CONTINUE elements.

**visit\_for** (*for\_*)

Implements traversing through FOR loops.

Can be overridden to allow modifying the passed in `for_` without calling `start_for()` or `end_for()` nor visiting body.

**visit\_for\_iteration** (*iteration*)

Implements traversing through single FOR loop iteration.

This is only used with the result side model because on the running side there are no iterations.

Can be overridden to allow modifying the passed in *iteration* without calling *start\_for\_iteration()* or *end\_for\_iteration()* nor visiting body.

**visit\_if** (*if\_*)

Implements traversing through IF/ELSE structures.

Notice that *if\_* does not have any data directly. Actual IF/ELSE branches are in its *body* and visited using *visit\_if\_branch()*.

Can be overridden to allow modifying the passed in *if\_* without calling *start\_if()* or *end\_if()* nor visiting branches.

**visit\_if\_branch** (*branch*)

Implements traversing through single IF/ELSE branch.

Can be overridden to allow modifying the passed in *branch* without calling *start\_if\_branch()* or *end\_if\_branch()* nor visiting body.

**visit\_message** (*msg*)

Implements visiting messages.

Can be overridden to allow modifying the passed in *msg* without calling *start\_message()* or *end\_message()*.

**visit\_return** (*return\_*)

Visits a RETURN elements.

**visit\_suite** (*suite*)

Implements traversing through suites.

Can be overridden to allow modifying the passed in *suite* without calling *start\_suite()* or *end\_suite()* nor visiting child suites, tests or setup and teardown at all.

**visit\_try** (*try\_*)

Implements traversing through TRY/EXCEPT structures.

This method is used with the TRY/EXCEPT root element. Actual TRY, EXCEPT, ELSE and FINALLY branches are visited separately using *visit\_try\_branch()*.

**visit\_try\_branch** (*branch*)

Visits individual TRY, EXCEPT, ELSE and FINALLY branches.

**visit\_while** (*while\_*)

Implements traversing through WHILE loops.

Can be overridden to allow modifying the passed in *while\_* without calling *start\_while()* or *end\_while()* nor visiting body.

**visit\_while\_iteration** (*iteration*)

Implements traversing through single WHILE loop iteration.

This is only used with the result side model because on the running side there are no iterations.

Can be overridden to allow modifying the passed in *iteration* without calling *start\_while\_iteration()* or *end\_while\_iteration()* nor visiting body.

**robot.model.tagstatistics module**

```

class robot.model.tagstatistics.TagStatistics (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.

    combined = None
        List of CombinedTagStat objects.

    visit (visitor)

class robot.model.tagstatistics.TagStatisticsBuilder (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_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, lineno=None, parent=None)
    Bases: robot.model.modelobject.ModelObject

    Base model for a single test case.

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

    body_class
        alias of robot.model.body.Body

    fixture_class
        alias of robot.model.keyword.Keyword

    repr_args = ('name',)

```

**name**

**doc**

**timeout**

**lineno**

**parent**

**body**

Test body as a *Body* object.

**tags**

Test tags as a *Tags* object.

**setup**

Test setup as a *Keyword* object.

This attribute is a *Keyword* object also when a test has no setup but in that case its truth value is `False`.

Setup can be modified by setting attributes directly:

```
test.setup.name = 'Example'
test.setup.args = ('First', 'Second')
```

Alternatively the *config()* method can be used to set multiple attributes in one call:

```
test.setup.config(name='Example', args=('First', 'Second'))
```

The easiest way to reset the whole setup is setting it to `None`. It will automatically recreate the underlying *Keyword* object:

```
test.setup = None
```

New in Robot Framework 4.0. Earlier setup was accessed like `test.keywords.setup`.

**has\_setup**

Check does a suite have a setup without creating a setup object.

A difference between using `if test.has_setup:` and `if test.setup:` is that accessing the *setup* attribute creates a *Keyword* object representing the setup even when the test actually does not have one. This typically does not matter, but with bigger suite structures containing a huge amount of tests it can have an effect on memory usage.

New in Robot Framework 5.0.

**teardown**

Test teardown as a *Keyword* object.

See *setup* for more information.

**has\_teardown**

Check does a test have a teardown without creating a teardown object.

See *has\_setup* for more information.

New in Robot Framework 5.0.

**keywords**

Deprecated since Robot Framework 4.0

Use *body*, *setup* or *teardown* instead.

**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.

**config** (*\*\*attributes*)

Configure model object with given attributes.

`obj.config(name='Example', doc='Something')` is equivalent to setting `obj.name = 'Example'` and `obj.doc = 'Something'`.

New in Robot Framework 4.0.

**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.

**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.

**class** `robot.model.testcase.TestCases` (*test\_class=<class 'robot.model.testcase.TestCase'>*,  
*parent=None, tests=None*)

Bases: `robot.model.itemlist.ItemList`

**append** (*item*)

`S.append(value)` – append value to the end of the sequence

**clear** () → None

– remove all items from S

**count** (*value*) → integer

– return number of occurrences of value

**create** (*\*args, \*\*kwargs*)**extend** (*items*)

`S.extend(iterable)` – extend sequence by appending elements from the iterable

**index** (*value* [, *start* [, *stop* ]]) → integer

– return first index of value.

Raises `ValueError` if the value is not present.

Supporting start and stop arguments is optional, but recommended.

**insert** (*index, item*)

`S.insert(index, value)` – insert value before index

**pop** ([*index*]) → item

– remove and return item at index (default last).

Raise `IndexError` if list is empty or index is out of range.

**remove** (*value*)  
S.remove(value) – remove first occurrence of value. Raise ValueError if the value is not present.

**reverse** ()  
S.reverse() – reverse *IN PLACE*

**sort** ()

**visit** (*visitor*)

## robot.model.testsuite module

[illegible]

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`

```
fixture_class
```

alias of `robot.model.keyword.Keyword`

```
repr_args = ('name',)
```

doc

**source**

Path to the source file or directory.

parent

Parent suite. None with the root suite.

rpa

True when RPA mode is enabled.

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.

setup

Suite setup as a *Keyword* object.

This attribute is a `Keyword` object also when a suite has no setup but in that case its truth value is `False`.

Setup can be modified by setting attributes directly:

```
suite.setup.name = 'Example'
suite.setup.args = ('First', 'Second')
```

Alternatively the `config()` method can be used to set multiple attributes in one call:

```
suite.setup.config(name='Example', args=('First', 'Second'))
```

The easiest way to reset the whole setup is setting it to `None`. It will automatically recreate the underlying `Keyword` object:

```
suite.setup = None
```

New in Robot Framework 4.0. Earlier setup was accessed like `suite.keywords.setup`.

### **has\_setup**

Check does a suite have a setup without creating a setup object.

A difference between using `if suite.has_setup:` and `if suite.setup:` is that accessing the `setup` attribute creates a `Keyword` object representing the setup even when the suite actually does not have one. This typically does not matter, but with bigger suite structures containing a huge amount of suites it can have some effect on memory usage.

New in Robot Framework 5.0.

### **teardown**

Suite teardown as a `Keyword` object.

See `setup` for more information.

### **has\_teardown**

Check does a suite have a teardown without creating a teardown object.

See `has_setup` for more information.

New in Robot Framework 5.0.

### **keywords**

Deprecated since Robot Framework 4.0

Use `setup` or `teardown` instead.

### **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')
```

**config** (*\*\*attributes*)

Configure model object with given attributes.

`obj.config(name='Example', doc='Something')` is equivalent to setting `obj.name = 'Example'` and `obj.doc = 'Something'`.

New in Robot Framework 4.0.

**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.

Not to be confused with *config()* method that suites, tests, and keywords have to make it possible to set multiple attributes in one call.

**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.

**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.

**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*)

`S.append(value)` – append value to the end of the sequence

**clear** () → None – remove all items from S



**count** (*value*) → integer – return number of occurrences of value

**create** (*\*args*, *\*\*kwargs*)

**extend** (*items*)  
 S.extend(iterable) – extend sequence by appending elements from the iterable

**index** (*value*[, *start*[, *stop*]]) → integer – return first index of value.  
 Raises ValueError if the value is not present.  
 Supporting start and stop arguments is optional, but recommended.

**insert** (*index*, *item*)  
 S.insert(index, value) – insert value before index

**pop** ([*index*]) → item – remove and return item at index (default last).  
 Raise IndexError if list is empty or index is out of range.

**remove** (*value*)  
 S.remove(value) – remove first occurrence of value. Raise ValueError if the value is not present.

**reverse** ()  
 S.reverse() – reverse *IN PLACE*

**sort** ()

**visit** (*visitor*)

### robot.model.totalstatistics module

**class** robot.model.totalstatistics.**TotalStatistics** (*rpa=False*)  
 Bases: object  
 Container for total statistics.

**visit** (*visitor*)

**total**

**passed**

**skipped**

**failed**

**add\_test** (*test*)

**message**  
 String representation of the statistics.  
 For example:: 2 tests, 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 tests.  
 Can be overridden to allow modifying the passed in test without calling *start\_test()* or *end\_test()* nor visiting the body of the test.

**visit\_keyword** (*kw*)

Implements traversing through keywords.

Can be overridden to allow modifying the passed in *kw* without calling `start_keyword()` or `end_keyword()` nor visiting the body of the keyword

**end\_body\_item** (*item*)

Called, by default, when keywords, messages or control structures end.

More specific `end_keyword()`, `end_message()`, `:meth:'end_for`, etc. can be implemented to visit only keywords, messages or specific control structures.

Default implementation does nothing.

**end\_break** (*break\_*)

Called when a BREAK element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_continue** (*continue\_*)

Called when a CONTINUE element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_for** (*for\_*)

Called when a FOR loop ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_for\_iteration** (*iteration*)

Called when a FOR loop iteration ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_if** (*if\_*)

Called when an IF/ELSE structure ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_if\_branch** (*branch*)

Called when an IF/ELSE branch ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_keyword** (*keyword*)

Called when a keyword ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_message** (*msg*)

Called when a message ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_return** (*return\_*)

Called when a RETURN element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_suite** (*suite*)

Called when a suite ends. Default implementation does nothing.

**end\_test** (*test*)

Called when a test ends. Default implementation does nothing.

**end\_try** (*try\_*)

Called when a TRY/EXCEPT structure ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_try\_branch** (*branch*)

Called when TRY, EXCEPT, ELSE and FINALLY branches end.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_while** (*while\_*)

Called when a WHILE loop ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_while\_iteration** (*iteration*)

Called when a WHILE loop iteration ends.

By default, calls `end_body_item()` which, by default, does nothing.

**start\_body\_item** (*item*)

Called, by default, when keywords, messages or control structures start.

More specific `start_keyword()`, `start_message()`, `:meth:'start_for`, etc. can be implemented to visit only keywords, messages or specific control structures.

Can return explicit `False` to stop visiting. Default implementation does nothing.

**start\_break** (*break\_*)

Called when a BREAK element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_continue** (*continue\_*)

Called when a CONTINUE element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_for** (*for\_*)

Called when a FOR loop starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_for\_iteration** (*iteration*)

Called when a FOR loop iteration starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_if** (*if\_*)

Called when an IF/ELSE structure starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_if\_branch** (*branch*)

Called when an IF/ELSE branch starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_keyword** (*keyword*)

Called when a keyword starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_message** (*msg*)

Called when a message starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_return** (*return\_*)

Called when a RETURN element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_suite** (*suite*)

Called when a suite starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

**start\_test** (*test*)

Called when a test starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

**start\_try** (*try\_*)

Called when a TRY/EXCEPT structure starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_try\_branch** (*branch*)

Called when TRY, EXCEPT, ELSE or FINALLY branches start.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_while** (*while\_*)

Called when a WHILE loop starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_while\_iteration** (*iteration*)

Called when a WHILE loop iteration starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**visit\_break** (*break\_*)

Visits BREAK elements.

**visit\_continue** (*continue\_*)

Visits CONTINUE elements.

**visit\_for** (*for\_*)

Implements traversing through FOR loops.

Can be overridden to allow modifying the passed in *for\_* without calling *start\_for()* or *end\_for()* nor visiting body.

**visit\_for\_iteration** (*iteration*)

Implements traversing through single FOR loop iteration.

This is only used with the result side model because on the running side there are no iterations.

Can be overridden to allow modifying the passed in *iteration* without calling *start\_for\_iteration()* or *end\_for\_iteration()* nor visiting body.

**visit\_if** (*if\_*)

Implements traversing through IF/ELSE structures.

Notice that *if\_* does not have any data directly. Actual IF/ELSE branches are in its body and visited using *visit\_if\_branch()*.

Can be overridden to allow modifying the passed in *if\_* without calling *start\_if()* or *end\_if()* nor visiting branches.

**visit\_if\_branch** (*branch*)

Implements traversing through single IF/ELSE branch.

Can be overridden to allow modifying the passed in *branch* without calling *start\_if\_branch()* or *end\_if\_branch()* nor visiting body.

**visit\_message** (*msg*)

Implements visiting messages.

Can be overridden to allow modifying the passed in *msg* without calling *start\_message()* or *end\_message()*.

**visit\_return** (*return\_*)

Visits a RETURN elements.

**visit\_suite** (*suite*)

Implements traversing through suites.

Can be overridden to allow modifying the passed in *suite* without calling *start\_suite()* or *end\_suite()* nor visiting child suites, tests or setup and teardown at all.

**visit\_try** (*try\_*)

Implements traversing through TRY/EXCEPT structures.

This method is used with the TRY/EXCEPT root element. Actual TRY, EXCEPT, ELSE and FINALLY branches are visited separately using *visit\_try\_branch()*.

**visit\_try\_branch** (*branch*)

Visits individual TRY, EXCEPT, ELSE and FINALLY branches.

**visit\_while** (*while\_*)

Implements traversing through WHILE loops.

Can be overridden to allow modifying the passed in *while\_* without calling *start\_while()* or *end\_while()* nor visiting body.

**visit\_while\_iteration** (*iteration*)

Implements traversing through single WHILE loop iteration.

This is only used with the result side model because on the running side there are no iterations.

Can be overridden to allow modifying the passed in iteration without calling `start_while_iteration()` or `end_while_iteration()` nor visiting body.

## 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 the 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.

All items that can appear inside tests have their own visit methods. These include `visit_keyword()`, `visit_message()` (only applicable with results, not with executable data), `visit_for()`, `visit_if()`, and so on, as well as their appropriate start/end methods like `start_keyword()` and `end_for()`. If there is a need to visit all these items, it is possible to implement only `start_body_item()` and `end_body_item()` methods that are, by default, called by the appropriate start/end methods. These generic methods are new in Robot Framework 5.0.

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.

```
"""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):
```

(continues on next page)

(continued from previous page)

```

def __init__(self, x: int, start: int = 0):
    self.x = x
    self.start = 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 suite structure.

See the *module level* documentation for more information and an example.

**visit\_suite** (*suite*)

Implements traversing through suites.

Can be overridden to allow modifying the passed in *suite* without calling *start\_suite()* or *end\_suite()* nor visiting child suites, tests or setup and teardown at all.

**start\_suite** (*suite*)

Called when a suite starts. Default implementation does nothing.

Can return explicit *False* to stop visiting.

**end\_suite** (*suite*)

Called when a suite ends. Default implementation does nothing.

**visit\_test** (*test*)

Implements traversing through tests.

Can be overridden to allow modifying the passed in *test* without calling *start\_test()* or *end\_test()* nor visiting the body of the test.

**start\_test** (*test*)

Called when a test starts. Default implementation does nothing.

Can return explicit *False* to stop visiting.

**end\_test** (*test*)

Called when a test ends. Default implementation does nothing.

**visit\_keyword** (*kw*)

Implements traversing through keywords.

Can be overridden to allow modifying the passed in *kw* without calling *start\_keyword()* or *end\_keyword()* nor visiting the body of the keyword

**start\_keyword** (*keyword*)

Called when a keyword starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**end\_keyword** (*keyword*)

Called when a keyword ends.

By default, calls `end_body_item()` which, by default, does nothing.

**visit\_for** (*for\_*)

Implements traversing through FOR loops.

Can be overridden to allow modifying the passed in *for\_* without calling `start_for()` or `end_for()` nor visiting body.

**start\_for** (*for\_*)

Called when a FOR loop starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**end\_for** (*for\_*)

Called when a FOR loop ends.

By default, calls `end_body_item()` which, by default, does nothing.

**visit\_for\_iteration** (*iteration*)

Implements traversing through single FOR loop iteration.

This is only used with the result side model because on the running side there are no iterations.

Can be overridden to allow modifying the passed in *iteration* without calling `start_for_iteration()` or `end_for_iteration()` nor visiting body.

**start\_for\_iteration** (*iteration*)

Called when a FOR loop iteration starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**end\_for\_iteration** (*iteration*)

Called when a FOR loop iteration ends.

By default, calls `end_body_item()` which, by default, does nothing.

**visit\_if** (*if\_*)

Implements traversing through IF/ELSE structures.

Notice that *if\_* does not have any data directly. Actual IF/ELSE branches are in its *body* and visited using `visit_if_branch()`.

Can be overridden to allow modifying the passed in *if\_* without calling `start_if()` or `end_if()` nor visiting branches.

**start\_if** (*if\_*)

Called when an IF/ELSE structure starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**end\_if** (*if\_*)

Called when an IF/ELSE structure ends.

By default, calls `end_body_item()` which, by default, does nothing.



**visit\_if\_branch** (*branch*)

Implements traversing through single IF/ELSE branch.

Can be overridden to allow modifying the passed in *branch* without calling `start_if_branch()` or `end_if_branch()` nor visiting body.

**start\_if\_branch** (*branch*)

Called when an IF/ELSE branch starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**end\_if\_branch** (*branch*)

Called when an IF/ELSE branch ends.

By default, calls `end_body_item()` which, by default, does nothing.

**visit\_try** (*try\_*)

Implements traversing through TRY/EXCEPT structures.

This method is used with the TRY/EXCEPT root element. Actual TRY, EXCEPT, ELSE and FINALLY branches are visited separately using `visit_try_branch()`.

**start\_try** (*try\_*)

Called when a TRY/EXCEPT structure starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**end\_try** (*try\_*)

Called when a TRY/EXCEPT structure ends.

By default, calls `end_body_item()` which, by default, does nothing.

**visit\_try\_branch** (*branch*)

Visits individual TRY, EXCEPT, ELSE and FINALLY branches.

**start\_try\_branch** (*branch*)

Called when TRY, EXCEPT, ELSE or FINALLY branches start.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**end\_try\_branch** (*branch*)

Called when TRY, EXCEPT, ELSE and FINALLY branches end.

By default, calls `end_body_item()` which, by default, does nothing.

**visit\_while** (*while\_*)

Implements traversing through WHILE loops.

Can be overridden to allow modifying the passed in *while\_* without calling `start_while()` or `end_while()` nor visiting body.

**start\_while** (*while\_*)

Called when a WHILE loop starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**end\_while** (*while\_*)

Called when a WHILE loop ends.

By default, calls `end_body_item()` which, by default, does nothing.

**visit\_while\_iteration** (*iteration*)

Implements traversing through single WHILE loop iteration.

This is only used with the result side model because on the running side there are no iterations.

Can be overridden to allow modifying the passed in *iteration* without calling `start_while_iteration()` or `end_while_iteration()` nor visiting body.

**start\_while\_iteration** (*iteration*)

Called when a WHILE loop iteration starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**end\_while\_iteration** (*iteration*)

Called when a WHILE loop iteration ends.

By default, calls `end_body_item()` which, by default, does nothing.

**visit\_return** (*return\_*)

Visits a RETURN elements.

**start\_return** (*return\_*)

Called when a RETURN element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**end\_return** (*return\_*)

Called when a RETURN element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**visit\_continue** (*continue\_*)

Visits CONTINUE elements.

**start\_continue** (*continue\_*)

Called when a CONTINUE element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**end\_continue** (*continue\_*)

Called when a CONTINUE element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**visit\_break** (*break\_*)

Visits BREAK elements.

**start\_break** (*break\_*)

Called when a BREAK element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**end\_break** (*break\_*)

Called when a BREAK element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**visit\_message** (*msg*)

Implements visiting messages.

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

**start\_message** (*msg*)

Called when a message starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**end\_message** (*msg*)

Called when a message ends.

By default, calls `end_body_item()` which, by default, does nothing.

**start\_body\_item** (*item*)

Called, by default, when keywords, messages or control structures start.

More specific `start_keyword()`, `start_message()`, `:meth:'start_for'`, etc. can be implemented to visit only keywords, messages or specific control structures.

Can return explicit `False` to stop visiting. Default implementation does nothing.

**end\_body\_item** (*item*)

Called, by default, when keywords, messages or control structures end.

More specific `end_keyword()`, `end_message()`, `:meth:'end_for'`, etc. can be implemented to visit only keywords, messages or specific control structures.

Default implementation does nothing.

**robot.output package**

Package for internal logging and other output.

Not part of the public API, and also subject to change in the future when test execution is refactored.

**Subpackages****robot.output.console package**

```
robot.output.console.ConsoleOutput (type='verbose', width=78, colors='AUTO', markers='AUTO', stdout=None, stderr=None)
```

**Submodules****robot.output.console.dotted module**

```
class robot.output.console.dotted.DottedOutput (width=78, colors='AUTO', stdout=None, stderr=None)
```

Bases: object

```
start_suite(suite)

end_test(test)

end_suite(suite)

message(msg)

output_file(name, path)
```

**class** `robot.output.console.dotted.StatusReporter` (*stream, width*)  
Bases: `robot.model.visitor.SuiteVisitor`

**report** (*suite*)

**visit\_test** (*test*)  
Implements traversing through tests.

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

**end\_body\_item** (*item*)  
Called, by default, when keywords, messages or control structures end.

More specific `end_keyword()`, `end_message()`, `:meth:'end_for'`, etc. can be implemented to visit only keywords, messages or specific control structures.

Default implementation does nothing.

**end\_break** (*break\_*)  
Called when a BREAK element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_continue** (*continue\_*)  
Called when a CONTINUE element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_for** (*for\_*)  
Called when a FOR loop ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_for\_iteration** (*iteration*)  
Called when a FOR loop iteration ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_if** (*if\_*)  
Called when an IF/ELSE structure ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_if\_branch** (*branch*)  
Called when an IF/ELSE branch ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_keyword** (*keyword*)  
Called when a keyword ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_message** (*msg*)  
Called when a message ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_return** (*return\_*)

Called when a RETURN element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_suite** (*suite*)

Called when a suite ends. Default implementation does nothing.

**end\_test** (*test*)

Called when a test ends. Default implementation does nothing.

**end\_try** (*try\_*)

Called when a TRY/EXCEPT structure ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_try\_branch** (*branch*)

Called when TRY, EXCEPT, ELSE and FINALLY branches end.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_while** (*while\_*)

Called when a WHILE loop ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_while\_iteration** (*iteration*)

Called when a WHILE loop iteration ends.

By default, calls `end_body_item()` which, by default, does nothing.

**start\_body\_item** (*item*)

Called, by default, when keywords, messages or control structures start.

More specific `start_keyword()`, `start_message()`, `:meth:'start_for`, etc. can be implemented to visit only keywords, messages or specific control structures.

Can return explicit `False` to stop visiting. Default implementation does nothing.

**start\_break** (*break\_*)

Called when a BREAK element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_continue** (*continue\_*)

Called when a CONTINUE element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_for** (*for\_*)

Called when a FOR loop starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_for\_iteration** (*iteration*)

Called when a FOR loop iteration starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_if** (*if\_*)

Called when an IF/ELSE structure starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_if\_branch** (*branch*)

Called when an IF/ELSE branch starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_keyword** (*keyword*)

Called when a keyword starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_message** (*msg*)

Called when a message starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_return** (*return\_*)

Called when a RETURN element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_suite** (*suite*)

Called when a suite starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

**start\_test** (*test*)

Called when a test starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

**start\_try** (*try\_*)

Called when a TRY/EXCEPT structure starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_try\_branch** (*branch*)

Called when TRY, EXCEPT, ELSE or FINALLY branches start.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_while** (*while\_*)

Called when a WHILE loop starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_while\_iteration** (*iteration*)

Called when a WHILE loop iteration starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**visit\_break** (*break\_*)

Visits BREAK elements.

**visit\_continue** (*continue\_*)

Visits CONTINUE elements.

**visit\_for** (*for\_*)

Implements traversing through FOR loops.

Can be overridden to allow modifying the passed in `for_` without calling `start_for()` or `end_for()` nor visiting body.

**visit\_for\_iteration** (*iteration*)

Implements traversing through single FOR loop iteration.

This is only used with the result side model because on the running side there are no iterations.

Can be overridden to allow modifying the passed in `iteration` without calling `start_for_iteration()` or `end_for_iteration()` nor visiting body.

**visit\_if** (*if\_*)

Implements traversing through IF/ELSE structures.

Notice that `if_` does not have any data directly. Actual IF/ELSE branches are in its body and visited using `visit_if_branch()`.

Can be overridden to allow modifying the passed in `if_` without calling `start_if()` or `end_if()` nor visiting branches.

**visit\_if\_branch** (*branch*)

Implements traversing through single IF/ELSE branch.

Can be overridden to allow modifying the passed in `branch` without calling `start_if_branch()` or `end_if_branch()` nor visiting body.

**visit\_keyword** (*kw*)

Implements traversing through keywords.

Can be overridden to allow modifying the passed in `kw` without calling `start_keyword()` or `end_keyword()` nor visiting the body of the keyword

**visit\_message** (*msg*)

Implements visiting messages.

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

**visit\_return** (*return\_*)

Visits a RETURN elements.

**visit\_suite** (*suite*)

Implements traversing through suites.

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

**visit\_try** (*try\_*)

Implements traversing through TRY/EXCEPT structures.

This method is used with the TRY/EXCEPT root element. Actual TRY, EXCEPT, ELSE and FINALLY branches are visited separately using *visit\_try\_branch()*.

**visit\_try\_branch** (*branch*)

Visits individual TRY, EXCEPT, ELSE and FINALLY branches.

**visit\_while** (*while\_*)

Implements traversing through WHILE loops.

Can be overridden to allow modifying the passed in *while\_* without calling *start\_while()* or *end\_while()* nor visiting body.

**visit\_while\_iteration** (*iteration*)

Implements traversing through single WHILE loop iteration.

This is only used with the result side model because on the running side there are no iterations.

Can be overridden to allow modifying the passed in *iteration* without calling *start\_while\_iteration()* or *end\_while\_iteration()* nor visiting body.

## robot.output.console.highlighting module

```
class robot.output.console.highlighting.HighlightingStream(stream, col-  
ors='AUTO')
```

Bases: object

**write** (*text*, *flush*=True)

**flush** ()

**highlight** (*text*, *status*=None, *flush*=True)

**error** (*message*, *level*)

```
robot.output.console.highlighting.Highlighter(stream)
```

```
class robot.output.console.highlighting.AnsiHighlighter(stream)
```

Bases: object

**green** ()

**red** ()

**yellow** ()

**reset** ()

```
class robot.output.console.highlighting.NoHighlighting(stream)
```

Bases: *robot.output.console.highlighting.AnsiHighlighter*

**green** ()

**red** ()

**reset** ()

**yellow** ()

```
class robot.output.console.highlighting.DosHighlighter(stream)
```

Bases: object

**green** ()



```

red()
yellow()
reset()

```

### robot.output.console.quiet module

```

class robot.output.console.quiet.QuietOutput (colors='AUTO', stderr=None)
    Bases: object

    message (msg)

class robot.output.console.quiet.NoOutput
    Bases: object

```

### robot.output.console.verbose module

```

class robot.output.console.verbose.VerboseOutput (width=78,          colors='AUTO',
                                                    markers='AUTO',    stdout=None,
                                                    stderr=None)

    Bases: object

    start_suite (suite)
    end_suite (suite)
    start_test (test)
    end_test (test)
    start_keyword (kw)
    end_keyword (kw)
    message (msg)
    output_file (name, path)

class robot.output.console.verbose.VerboseWriter (width=78,          colors='AUTO',
                                                    markers='AUTO',    stdout=None,
                                                    stderr=None)

    Bases: object

    info (name, doc, start_suite=False)
    suite_separator ()
    test_separator ()
    status (status, clear=False)
    message (message)
    keyword_marker (status)
    error (message, level, clear=False)
    output (name, path)

class robot.output.console.verbose.KeywordMarker (highlighter, markers)
    Bases: object

    mark (status)

```

```
reset_count ()
```

## Submodules

### robot.output.debugfile module

```
robot.output.debugfile.DebugFile (path)
```

### robot.output.filelogger module

```
class robot.output.filelogger.FileLogger (path, level)
    Bases: robot.output.loggerhelper.AbstractLogger
    message (msg)
    start_suite (suite)
    end_suite (suite)
    start_test (test)
    end_test (test)
    start_keyword (kw)
    end_keyword (kw)
    output_file (name, path)
    close ()
    debug (msg)
    error (msg)
    fail (msg)
    info (msg)
    set_level (level)
    skip (msg)
    trace (msg)
    warn (msg)
    write (message, level, html=False)
```

### robot.output.librarylogger module

Implementation of the public logging API for libraries.

This is exposed via *robot.api.logger*. Implementation must reside here to avoid cyclic imports.

```
robot.output.librarylogger.write (msg, level, html=False)
```

```
robot.output.librarylogger.trace (msg, html=False)
```

```
robot.output.librarylogger.debug (msg, html=False)
```

```
robot.output.librarylogger.info (msg, html=False, also_console=False)
```

```
robot.output.librarylogger.warn (msg, html=False)
robot.output.librarylogger.error (msg, html=False)
robot.output.librarylogger.console (msg, newline=True, stream='stdout')
```

### robot.output.listenerarguments module

```
class robot.output.listenerarguments.ListenerArguments (arguments)
    Bases: object
        get_arguments (version)
        classmethod by_method_name (name, arguments)

class robot.output.listenerarguments.MessageArguments (arguments)
    Bases: robot.output.listenerarguments.ListenerArguments
        classmethod by_method_name (name, arguments)
        get_arguments (version)

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

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

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

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

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

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

**robot.output.listenermethods module**

```
class robot.output.listenermethods.ListenerMethods (method_name, listeners)
    Bases: object

class robot.output.listenermethods.LibraryListenerMethods (method_name)
    Bases: object

    new_suite_scope ()

    discard_suite_scope ()

    register (listeners, library)

    unregister (library)

class robot.output.listenermethods.ListenerMethod (method, listener, library=None)
    Bases: object

    called = False
```

**robot.output.listeners module**

```
class robot.output.listeners.Listeners (listeners, log_level='INFO')
    Bases: object

    set_log_level (level)

    start_keyword (kw)

    end_keyword (kw)

    log_message (msg)

    imported (import_type, name, attrs)

    output_file (file_type, path)

class robot.output.listeners.LibraryListeners (log_level='INFO')
    Bases: object

    register (listeners, library)

    unregister (library, close=False)

    new_suite_scope ()

    discard_suite_scope ()

    set_log_level (level)

    log_message (msg)

    imported (import_type, name, attrs)

    output_file (file_type, path)

class robot.output.listeners.ListenerProxy (listener, method_names, prefix=None)
    Bases: robot.output.loggerhelper.AbstractLoggerProxy

    classmethod import_listeners (listeners, method_names, prefix=None, raise_on_error=False)
```

**robot.output.logger module**

**class** `robot.output.logger.Logger` (*register\_console\_logger=True*)

Bases: `robot.output.loggerhelper.AbstractLogger`

A global logger proxy to delegating messages to registered loggers.

Whenever something is written to `LOGGER` in code, all registered loggers are notified. Messages are also cached and cached messages written to new loggers when they are registered.

NOTE: This API is likely to change in future versions.

**start\_loggers**

**end\_loggers**

**register\_console\_logger** (*type='verbose', width=78, colors='AUTO', markers='AUTO', stdout=None, stderr=None*)

**unregister\_console\_logger** ()

**register\_syslog** (*path=None, level='INFO'*)

**register\_xml\_logger** (*logger*)

**unregister\_xml\_logger** ()

**register\_listeners** (*listeners, library\_listeners*)

**register\_logger** (*\*loggers*)

**unregister\_logger** (*\*loggers*)

**disable\_message\_cache** ()

**register\_error\_listener** (*listener*)

**message** (*msg*)

Messages about what the framework is doing, warnings, errors, ...

**cache\_only**

**delayed\_logging**

**log\_message** (*msg*)

Messages about what the framework is doing, warnings, errors, ...

**log\_output** (*output*)

**enable\_library\_import\_logging** ()

**disable\_library\_import\_logging** ()

**start\_suite** (*suite*)

**end\_suite** (*suite*)

**start\_test** (*test*)

**end\_test** (*test*)

**start\_keyword** (*keyword*)

**end\_keyword** (*keyword*)

**imported** (*import\_type, name, \*\*attrs*)

**output\_file** (*file\_type, path*)

Finished output, report, log, debug, or xunit file

```
close()
debug(msg)
error(msg)
fail(msg)
info(msg)
set_level(level)
skip(msg)
trace(msg)
warn(msg)
write(message, level, html=False)

class robot.output.logger.LoggerProxy(logger, method_names=None, prefix=None)
    Bases: robot.output.loggerhelper.AbstractLoggerProxy
    start_keyword(kw)
    end_keyword(kw)
```

### 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)
    skip(msg)
    error(msg)
    write(message, level, html=False)
    message(msg)

class robot.output.loggerhelper.Message(message, level='INFO', html=False, times-
    tamp=None)
    Bases: robot.model.message.Message
    message
    resolve_delayed_message()
    BREAK = 'BREAK'
    CONTINUE = 'CONTINUE'
    ELSE = 'ELSE'
    ELSE_IF = 'ELSE IF'
```

```

EXCEPT = 'EXCEPT'
FINALLY = 'FINALLY'
FOR = 'FOR'
IF = 'IF'
IF_ELSE_ROOT = 'IF/ELSE ROOT'
ITERATION = 'ITERATION'
KEYWORD = 'KEYWORD'
MESSAGE = 'MESSAGE'
RETURN = 'RETURN'
SETUP = 'SETUP'
TEARDOWN = 'TEARDOWN'
TRY = 'TRY'
TRY_EXCEPT_ROOT = 'TRY/EXCEPT ROOT'
WHILE = 'WHILE'

```

**config** (*\*\*attributes*)

Configure model object with given attributes.

`obj.config(name='Example', doc='Something')` is equivalent to setting `obj.name = 'Example'` and `obj.doc = 'Something'`.

New in Robot Framework 4.0.

**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.

**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.

**has\_setup**

**has\_teardown**

**html**

**html\_message**

Returns the message content as HTML.

**id**

Item id in format like `s1-t3-k1`.

See [TestSuite.id](#) for more information.

```
    level
    parent
    repr_args = ('message', 'level')
    timestamp
    type = 'MESSAGE'
    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)
    skip(msg)
    trace(msg)
    warn(msg)
    write(message, level, html=False)
```



**robot.output.pyloggingconf module**

`robot.output.pyloggingconf.robot_handler_enabled(level)`

`robot.output.pyloggingconf.set_level(level)`

```
class robot.output.pyloggingconf.RobotHandler (level=0, library_logger=<module
'robot.output.librarylogger' from
'/home/docs/checkouts/readthedocs.org/user_builds/robot-
framework/checkouts/v5.0.1/src/robot/output/librarylogger.py'>)
```

Bases: `logging.Handler`

**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.

Changed in version 3.2: Allow filters to be just callables.

**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. level must be an int or a str.  
**set\_name** (*name*)

### robot.output.stdoutlogsplitter module

**class** robot.output.stdoutlogsplitter.StdoutLogSplitter(*output*)  
Bases: object  
Splits messages logged through stdout (or stderr) into Message objects

### robot.output.xmllogger module

**class** robot.output.xmllogger.XmlLogger(*path*, *log\_level*=*'TRACE'*, *rpa*=*False*, *generator*=*'Robot'*)  
Bases: *robot.result.visitor.ResultVisitor*  
**close()**  
**set\_log\_level** (*level*)  
**message** (*msg*)  
**log\_message** (*msg*)  
**start\_keyword** (*kw*)  
Called when a keyword starts.  
By default, calls *start\_body\_item()* which, by default, does nothing.  
Can return explicit *False* to stop visiting.  
**end\_keyword** (*kw*)  
Called when a keyword ends.  
By default, calls *end\_body\_item()* which, by default, does nothing.  
**start\_if** (*if\_*)  
Called when an IF/ELSE structure starts.  
By default, calls *start\_body\_item()* which, by default, does nothing.  
Can return explicit *False* to stop visiting.

**end\_if** (*if\_*)

Called when an IF/ELSE structure ends.

By default, calls `end_body_item()` which, by default, does nothing.

**start\_if\_branch** (*branch*)

Called when an IF/ELSE branch starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**end\_if\_branch** (*branch*)

Called when an IF/ELSE branch ends.

By default, calls `end_body_item()` which, by default, does nothing.

**start\_for** (*for\_*)

Called when a FOR loop starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**end\_for** (*for\_*)

Called when a FOR loop ends.

By default, calls `end_body_item()` which, by default, does nothing.

**start\_for\_iteration** (*iteration*)

Called when a FOR loop iteration starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**end\_for\_iteration** (*iteration*)

Called when a FOR loop iteration ends.

By default, calls `end_body_item()` which, by default, does nothing.

**start\_try** (*root*)

Called when a TRY/EXCEPT structure starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**end\_try** (*root*)

Called when a TRY/EXCEPT structure ends.

By default, calls `end_body_item()` which, by default, does nothing.

**start\_try\_branch** (*branch*)

Called when TRY, EXCEPT, ELSE or FINALLY branches start.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**end\_try\_branch** (*branch*)

Called when TRY, EXCEPT, ELSE and FINALLY branches end.

By default, calls `end_body_item()` which, by default, does nothing.

**start\_while** (*while\_*)

Called when a WHILE loop starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**end\_while** (*while\_*)

Called when a WHILE loop ends.

By default, calls `end_body_item()` which, by default, does nothing.

**start\_while\_iteration** (*iteration*)

Called when a WHILE loop iteration starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**end\_while\_iteration** (*iteration*)

Called when a WHILE loop iteration ends.

By default, calls `end_body_item()` which, by default, does nothing.

**start\_return** (*return\_*)

Called when a RETURN element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**end\_return** (*return\_*)

Called when a RETURN element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**start\_continue** (*continue\_*)

Called when a CONTINUE element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**end\_continue** (*continue\_*)

Called when a CONTINUE element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**start\_break** (*break\_*)

Called when a BREAK element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**end\_break** (*break\_*)

Called when a BREAK element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**start\_test** (*test*)

Called when a test starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

**end\_test** (*test*)

Called when a test ends. Default implementation does nothing.

**start\_suite** (*suite*)

Called when a suite starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

**end\_suite** (*suite*)

Called when a 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\_body\_item** (*item*)

Called, by default, when keywords, messages or control structures end.

More specific `end_keyword()`, `end_message()`, `:meth:'end_for'`, etc. can be implemented to visit only keywords, messages or specific control structures.

Default implementation does nothing.

**end\_message** (*msg*)

Called when a message ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_result** (*result*)

**end\_stat** (*stat*)

**start\_body\_item** (*item*)

Called, by default, when keywords, messages or control structures start.

More specific `start_keyword()`, `start_message()`, `:meth:'start_for'`, etc. can be implemented to visit only keywords, messages or specific control structures.

Can return explicit `False` to stop visiting. Default implementation does nothing.

**start\_message** (*msg*)

Called when a message starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_result** (*result*)

**start\_stat** (*stat*)

**visit\_break** (*break\_*)

Visits `BREAK` elements.

**visit\_continue** (*continue\_*)  
Visits CONTINUE elements.

**visit\_errors** (*errors*)

**visit\_for** (*for\_*)  
Implements traversing through FOR loops.

Can be overridden to allow modifying the passed in *for\_* without calling *start\_for()* or *end\_for()* nor visiting body.

**visit\_for\_iteration** (*iteration*)  
Implements traversing through single FOR loop iteration.

This is only used with the result side model because on the running side there are no iterations.

Can be overridden to allow modifying the passed in *iteration* without calling *start\_for\_iteration()* or *end\_for\_iteration()* nor visiting body.

**visit\_if** (*if\_*)  
Implements traversing through IF/ELSE structures.

Notice that *if\_* does not have any data directly. Actual IF/ELSE branches are in its body and visited using *visit\_if\_branch()*.

Can be overridden to allow modifying the passed in *if\_* without calling *start\_if()* or *end\_if()* nor visiting branches.

**visit\_if\_branch** (*branch*)  
Implements traversing through single IF/ELSE branch.

Can be overridden to allow modifying the passed in *branch* without calling *start\_if\_branch()* or *end\_if\_branch()* nor visiting body.

**visit\_keyword** (*kw*)  
Implements traversing through keywords.

Can be overridden to allow modifying the passed in *kw* without calling *start\_keyword()* or *end\_keyword()* nor visiting the body of the keyword

**visit\_message** (*msg*)  
Implements visiting messages.

Can be overridden to allow modifying the passed in *msg* without calling *start\_message()* or *end\_message()*.

**visit\_result** (*result*)

**visit\_return** (*return\_*)  
Visits a RETURN elements.

**visit\_statistics** (*stats*)

**visit\_suite** (*suite*)  
Implements traversing through suites.

Can be overridden to allow modifying the passed in *suite* without calling *start\_suite()* or *end\_suite()* nor visiting child suites, tests or setup and teardown at all.

**visit\_suite\_statistics** (*stats*)

**visit\_tag\_statistics** (*stats*)

**visit\_test** (*test*)  
Implements traversing through tests.

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

**visit\_total\_statistics** (*stats*)

**visit\_try** (*try\_*)

Implements traversing through TRY/EXCEPT structures.

This method is used with the TRY/EXCEPT root element. Actual TRY, EXCEPT, ELSE and FINALLY branches are visited separately using `visit_try_branch()`.

**visit\_try\_branch** (*branch*)

Visits individual TRY, EXCEPT, ELSE and FINALLY branches.

**visit\_while** (*while\_*)

Implements traversing through WHILE loops.

Can be overridden to allow modifying the passed in `while_` without calling `start_while()` or `end_while()` nor visiting body.

**visit\_while\_iteration** (*iteration*)

Implements traversing through single WHILE loop iteration.

This is only used with the result side model because on the running side there are no iterations.

Can be overridden to allow modifying the passed in `iteration` without calling `start_while_iteration()` or `end_while_iteration()` nor visiting body.

## robot.parsing package

Module implementing test data parsing.

Public API is exposed via the `robot.api.parsing` module. See its documentation for more information and examples. If external code needs to import something from this module directly, issue should be submitted about exposing it explicitly via `robot.api.parsing`.

## Subpackages

### robot.parsing.lexer package

## Submodules

### robot.parsing.lexer.blocklexers module

**class** `robot.parsing.lexer.blocklexers.BlockLexer` (*ctx*)

Bases: `robot.parsing.lexer.statemlexers.Lexer`

**accepts\_more** (*statement*)

**input** (*statement*)

**lexer\_for** (*statement*)

**lexer\_classes** ()

**lex** ()

**handles** (*statement*)

```
class robot.parsing.lexer.blocklexers.FileLexer(ctx)
    Bases: robot.parsing.lexer.blocklexers.BlockLexer

    lex()
    lexer_classes()
    accepts_more(statement)
    handles(statement)
    input(statement)
    lexer_for(statement)

class robot.parsing.lexer.blocklexers.SectionLexer(ctx)
    Bases: robot.parsing.lexer.blocklexers.BlockLexer

    accepts_more(statement)
    handles(statement)
    input(statement)
    lex()
    lexer_classes()
    lexer_for(statement)

class robot.parsing.lexer.blocklexers.SettingSectionLexer(ctx)
    Bases: robot.parsing.lexer.blocklexers.SectionLexer

    handles(statement)
    lexer_classes()
    accepts_more(statement)
    input(statement)
    lex()
    lexer_for(statement)

class robot.parsing.lexer.blocklexers.VariableSectionLexer(ctx)
    Bases: robot.parsing.lexer.blocklexers.SectionLexer

    handles(statement)
    lexer_classes()
    accepts_more(statement)
    input(statement)
    lex()
    lexer_for(statement)

class robot.parsing.lexer.blocklexers.TestCaseSectionLexer(ctx)
    Bases: robot.parsing.lexer.blocklexers.SectionLexer
```



```
    handles (statement)
    lexer_classes ()
    accepts_more (statement)
    input (statement)
    lex ()
    lexer_for (statement)

class robot.parsing.lexer.blocklexers.KeywordSectionLexer (ctx)
    Bases: robot.parsing.lexer.blocklexers.SettingSectionLexer

    handles (statement)
    lexer_classes ()
    accepts_more (statement)
    input (statement)
    lex ()
    lexer_for (statement)

class robot.parsing.lexer.blocklexers.CommentSectionLexer (ctx)
    Bases: robot.parsing.lexer.blocklexers.SectionLexer

    handles (statement)
    lexer_classes ()
    accepts_more (statement)
    input (statement)
    lex ()
    lexer_for (statement)

class robot.parsing.lexer.blocklexers.ImplicitCommentSectionLexer (ctx)
    Bases: robot.parsing.lexer.blocklexers.SectionLexer

    handles (statement)
    lexer_classes ()
    accepts_more (statement)
    input (statement)
    lex ()
    lexer_for (statement)

class robot.parsing.lexer.blocklexers.ErrorSectionLexer (ctx)
    Bases: robot.parsing.lexer.blocklexers.SectionLexer
```

```
    handles (statement)
    lexer_classes ()
    accepts_more (statement)
    input (statement)
    lex ()
    lexer_for (statement)

class robot.parsing.lexer.blocklexers.TestOrKeywordLexer (ctx)
    Bases: robot.parsing.lexer.blocklexers.BlockLexer

    name_type = NotImplemented
    accepts_more (statement)
    input (statement)
    lexer_classes ()
    handles (statement)
    lex ()
    lexer_for (statement)

class robot.parsing.lexer.blocklexers.TestCaseLexer (ctx)
    Bases: robot.parsing.lexer.blocklexers.TestOrKeywordLexer

    name_type = 'TESTCASE NAME'
    lex ()
    accepts_more (statement)
    handles (statement)
    input (statement)
    lexer_classes ()
    lexer_for (statement)

class robot.parsing.lexer.blocklexers.KeywordLexer (ctx)
    Bases: robot.parsing.lexer.blocklexers.TestOrKeywordLexer

    name_type = 'KEYWORD NAME'
    accepts_more (statement)
    handles (statement)
    input (statement)
    lex ()
    lexer_classes ()
    lexer_for (statement)

class robot.parsing.lexer.blocklexers.NestedBlockLexer (ctx)
    Bases: robot.parsing.lexer.blocklexers.BlockLexer
```

```
    accepts_more (statement)
    input (statement)
    handles (statement)
    lex ()
    lexer_classes ()
    lexer_for (statement)

class robot.parsing.lexer.blocklexers.ForLexer (ctx)
    Bases: robot.parsing.lexer.blocklexers.NestedBlockLexer
    handles (statement)
    lexer_classes ()
    accepts_more (statement)
    input (statement)
    lex ()
    lexer_for (statement)

class robot.parsing.lexer.blocklexers.WhileLexer (ctx)
    Bases: robot.parsing.lexer.blocklexers.NestedBlockLexer
    handles (statement)
    lexer_classes ()
    accepts_more (statement)
    input (statement)
    lex ()
    lexer_for (statement)

class robot.parsing.lexer.blocklexers.IfLexer (ctx)
    Bases: robot.parsing.lexer.blocklexers.NestedBlockLexer
    handles (statement)
    lexer_classes ()
    accepts_more (statement)
    input (statement)
    lex ()
    lexer_for (statement)

class robot.parsing.lexer.blocklexers.InlineIfLexer (ctx)
    Bases: robot.parsing.lexer.blocklexers.BlockLexer

    handles (statement)
    accepts_more (statement)
    lexer_classes ()
    input (statement)
```

```
lex()
lexer_for(statement)
class robot.parsing.lexer.blocklexers.TryLexer(ctx)
    Bases: robot.parsing.lexer.blocklexers.NestedBlockLexer
    handles(statement)
    lexer_classes()
    accepts_more(statement)
    input(statement)
    lex()
    lexer_for(statement)
```

### robot.parsing.lexer.context module

```
class robot.parsing.lexer.context.LexingContext(settings=None)
    Bases: object
    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.lexer module

```
robot.parsing.lexer.lexer.get_tokens (source, data_only=False, tokenize_variables=False)
    Parses the given source to tokens.
```

### Parameters

- **source** – The source where to read the data. Can be a path to a source file as a string or as `pathlib.Path` object, an already opened file object, or Unicode text containing the data directly. Source files must be UTF-8 encoded.
- **data\_only** – When `False` (default), returns all tokens. When set to `True`, omits separators, comments, continuation markers, and other non-data tokens.
- **tokenize\_variables** – When `True`, possible variables in keyword arguments and elsewhere are tokenized. See the `tokenize_variables()` method for details.

Returns a generator that yields *Token* instances.

```
robot.parsing.lexer.lexer.get_resource_tokens (source, data_only=False, tokenize_variables=False)
```

Parses the given source to resource file tokens.

Otherwise same as `get_tokens()` but the source is considered to be a resource file. This affects, for example, what settings are valid.

```
robot.parsing.lexer.lexer.get_init_tokens (source, data_only=False, tokenize_variables=False)
```

Parses the given source to init file tokens.

Otherwise same as `get_tokens()` but the source is considered to be a suite initialization file. This affects, for example, what settings are valid.

```
class robot.parsing.lexer.lexer.Lexer (ctx, data_only=False, tokenize_variables=False)
    Bases: object

    input (source)

    get_tokens ()
```

## robot.parsing.lexer.sections module

```
class robot.parsing.lexer.sections.Sections
    Bases: object

    setting_markers = ('Settings', 'Setting')
    variable_markers = ('Variables', 'Variable')
    test_case_markers = ('Test Cases', 'Test Case', 'Tasks', 'Task')
    keyword_markers = ('Keywords', 'Keyword')
```

```
comment_markers = ('Comments', 'Comment')
setting(statement)
variable(statement)
test_case(statement)
keyword(statement)
comment(statement)
lex_invalid(statement)

class robot.parsing.lexer.sections.TestCaseFileSections
    Bases: robot.parsing.lexer.sections.Sections
    test_case(statement)
    comment(statement)
    comment_markers = ('Comments', 'Comment')
    keyword(statement)
    keyword_markers = ('Keywords', 'Keyword')
    lex_invalid(statement)
    setting(statement)
    setting_markers = ('Settings', 'Setting')
    test_case_markers = ('Test Cases', 'Test Case', 'Tasks', 'Task')
    variable(statement)
    variable_markers = ('Variables', 'Variable')

class robot.parsing.lexer.sections.ResourceFileSections
    Bases: robot.parsing.lexer.sections.Sections
    comment(statement)
    comment_markers = ('Comments', 'Comment')
    keyword(statement)
    keyword_markers = ('Keywords', 'Keyword')
    lex_invalid(statement)
    setting(statement)
    setting_markers = ('Settings', 'Setting')
    test_case(statement)
    test_case_markers = ('Test Cases', 'Test Case', 'Tasks', 'Task')
    variable(statement)
    variable_markers = ('Variables', 'Variable')

class robot.parsing.lexer.sections.InitFileSections
    Bases: robot.parsing.lexer.sections.Sections
    comment(statement)
    comment_markers = ('Comments', 'Comment')
```

```
keyword(statement)
keyword_markers = ('Keywords', 'Keyword')
lex_invalid(statement)
setting(statement)
setting_markers = ('Settings', 'Setting')
test_case(statement)
test_case_markers = ('Test Cases', 'Test Case', 'Tasks', 'Task')
variable(statement)
variable_markers = ('Variables', 'Variable')
```

### 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
    name_arguments_and_with_name = ('Library',)
    lex(statement)

class robot.parsing.lexer.settings.TestCaseFileSettings
    Bases: robot.parsing.lexer.settings.Settings
    names = ('Documentation', 'Metadata', 'Suite Setup', 'Suite Teardown', 'Test Setup', '
    aliases = {'Task Setup': 'Test Setup', 'Task Teardown': 'Test Teardown', 'Task Templ
    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.InitFileSettings
    Bases: robot.parsing.lexer.settings.Settings
    names = ('Documentation', 'Metadata', 'Suite Setup', 'Suite Teardown', 'Test Setup', '
    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.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
    name_arguments_and_with_name = ('Library',)
    single_value = ('Resource', 'Test Timeout', 'Test Template', 'Timeout', 'Template')
class robot.parsing.lexer.settings.TestCaseSettings(parent)
    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.statementlexers module

```

class robot.parsing.lexer.statementlexers.Lexer(ctx)
    Bases: object
    Base class for lexers.
    handles(statement)
    accepts_more(statement)
    input(statement)

```

```
lex()

class robot.parsing.lexer.statemlexers.StatementLexer(ctx)
    Bases: robot.parsing.lexer.statemlexers.Lexer
    token_type = None
    accepts_more(statement)
    input(statement)
    lex()
    handles(statement)

class robot.parsing.lexer.statemlexers.SingleType(ctx)
    Bases: robot.parsing.lexer.statemlexers.StatementLexer
    lex()
    accepts_more(statement)
    handles(statement)
    input(statement)
    token_type = None

class robot.parsing.lexer.statemlexers.TypeAndArguments(ctx)
    Bases: robot.parsing.lexer.statemlexers.StatementLexer
    lex()
    accepts_more(statement)
    handles(statement)
    input(statement)
    token_type = None

class robot.parsing.lexer.statemlexers.SectionHeaderLexer(ctx)
    Bases: robot.parsing.lexer.statemlexers.SingleType
    handles(statement)
    accepts_more(statement)
    input(statement)
    lex()
    token_type = None

class robot.parsing.lexer.statemlexers.SettingSectionHeaderLexer(ctx)
    Bases: robot.parsing.lexer.statemlexers.SectionHeaderLexer
    token_type = 'SETTING HEADER'
    accepts_more(statement)
    handles(statement)
    input(statement)
    lex()

class robot.parsing.lexer.statemlexers.VariableSectionHeaderLexer(ctx)
    Bases: robot.parsing.lexer.statemlexers.SectionHeaderLexer
```

```
    token_type = 'VARIABLE HEADER'
    accepts_more(statement)
    handles(statement)
    input(statement)
    lex()

class robot.parsing.lexer.statemlexers.TestCaseSectionHeaderLexer(ctx)
    Bases: robot.parsing.lexer.statemlexers.SectionHeaderLexer
    token_type = 'TESTCASE HEADER'
    accepts_more(statement)
    handles(statement)
    input(statement)
    lex()

class robot.parsing.lexer.statemlexers.KeywordSectionHeaderLexer(ctx)
    Bases: robot.parsing.lexer.statemlexers.SectionHeaderLexer
    token_type = 'KEYWORD HEADER'
    accepts_more(statement)
    handles(statement)
    input(statement)
    lex()

class robot.parsing.lexer.statemlexers.CommentSectionHeaderLexer(ctx)
    Bases: robot.parsing.lexer.statemlexers.SectionHeaderLexer
    token_type = 'COMMENT HEADER'
    accepts_more(statement)
    handles(statement)
    input(statement)
    lex()

class robot.parsing.lexer.statemlexers.ErrorSectionHeaderLexer(ctx)
    Bases: robot.parsing.lexer.statemlexers.SectionHeaderLexer
    lex()
    accepts_more(statement)
    handles(statement)
    input(statement)
    token_type = None

class robot.parsing.lexer.statemlexers.CommentLexer(ctx)
    Bases: robot.parsing.lexer.statemlexers.SingleType
    token_type = 'COMMENT'
    accepts_more(statement)
    handles(statement)
```

```
    input (statement)

    lex ()

class robot.parsing.lexer.statements.lexers.SettingLexer (ctx)
    Bases: robot.parsing.lexer.statements.lexers.StatementLexer

    lex ()

    accepts_more (statement)

    handles (statement)

    input (statement)

    token_type = None

class robot.parsing.lexer.statements.lexers.TestOrKeywordSettingLexer (ctx)
    Bases: robot.parsing.lexer.statements.lexers.SettingLexer

    handles (statement)

    accepts_more (statement)

    input (statement)

    lex ()

    token_type = None

class robot.parsing.lexer.statements.lexers.VariableLexer (ctx)
    Bases: robot.parsing.lexer.statements.lexers.TypeAndArguments

    token_type = 'VARIABLE'

    accepts_more (statement)

    handles (statement)

    input (statement)

    lex ()

class robot.parsing.lexer.statements.lexers.KeywordCallLexer (ctx)
    Bases: robot.parsing.lexer.statements.lexers.StatementLexer

    lex ()

    accepts_more (statement)

    handles (statement)

    input (statement)

    token_type = None

class robot.parsing.lexer.statements.lexers.ForHeaderLexer (ctx)
    Bases: robot.parsing.lexer.statements.lexers.StatementLexer

    separators = ('IN', 'IN RANGE', 'IN ENUMERATE', 'IN ZIP')

    handles (statement)

    lex ()

    accepts_more (statement)

    input (statement)

    token_type = None
```

```
class robot.parsing.lexer.statemlexers.IfHeaderLexer(ctx)
    Bases: robot.parsing.lexer.statemlexers.TypeAndArguments
        token_type = 'IF'
        handles(statement)
        accepts_more(statement)
        input(statement)
        lex()

class robot.parsing.lexer.statemlexers.InlineIfHeaderLexer(ctx)
    Bases: robot.parsing.lexer.statemlexers.StatementLexer
        token_type = 'INLINE IF'
        handles(statement)
        lex()
        accepts_more(statement)
        input(statement)

class robot.parsing.lexer.statemlexers.ElseIfHeaderLexer(ctx)
    Bases: robot.parsing.lexer.statemlexers.TypeAndArguments
        token_type = 'ELSE IF'
        handles(statement)
        accepts_more(statement)
        input(statement)
        lex()

class robot.parsing.lexer.statemlexers.ElseHeaderLexer(ctx)
    Bases: robot.parsing.lexer.statemlexers.TypeAndArguments
        token_type = 'ELSE'
        handles(statement)
        accepts_more(statement)
        input(statement)
        lex()

class robot.parsing.lexer.statemlexers.TryHeaderLexer(ctx)
    Bases: robot.parsing.lexer.statemlexers.TypeAndArguments
        token_type = 'TRY'
        handles(statement)
        accepts_more(statement)
        input(statement)
        lex()

class robot.parsing.lexer.statemlexers.ExceptHeaderLexer(ctx)
    Bases: robot.parsing.lexer.statemlexers.StatementLexer
        token_type = 'EXCEPT'
```

```
    handles (statement)
    lex ()
    accepts_more (statement)
    input (statement)

class robot.parsing.lexer.statements.lexers.FinallyHeaderLexer (ctx)
    Bases: robot.parsing.lexer.statements.lexers.TypeAndArguments
    token_type = 'FINALLY'
    handles (statement)
    accepts_more (statement)
    input (statement)
    lex ()

class robot.parsing.lexer.statements.lexers.WhileHeaderLexer (ctx)
    Bases: robot.parsing.lexer.statements.lexers.StatementLexer
    token_type = 'WHILE'
    handles (statement)
    lex ()
    accepts_more (statement)
    input (statement)

class robot.parsing.lexer.statements.lexers.EndLexer (ctx)
    Bases: robot.parsing.lexer.statements.lexers.TypeAndArguments
    token_type = 'END'
    handles (statement)
    accepts_more (statement)
    input (statement)
    lex ()

class robot.parsing.lexer.statements.lexers.ReturnLexer (ctx)
    Bases: robot.parsing.lexer.statements.lexers.TypeAndArguments
    token_type = 'RETURN STATEMENT'
    handles (statement)
    accepts_more (statement)
    input (statement)
    lex ()

class robot.parsing.lexer.statements.lexers.ContinueLexer (ctx)
    Bases: robot.parsing.lexer.statements.lexers.TypeAndArguments
    token_type = 'CONTINUE'
    handles (statement)
    accepts_more (statement)
    input (statement)
```

```

lex()

class robot.parsing.lexer.statemlexers.BreakLexer(ctx)
    Bases: robot.parsing.lexer.statemlexers.TypeAndArguments

    token_type = 'BREAK'

    handles(statement)

    accepts_more(statement)

    input(statement)

    lex()

```

### robot.parsing.lexer.tokenizer module

```

class robot.parsing.lexer.tokenizer.Tokenizer
    Bases: object

    tokenize(data, data_only=False)

```

### robot.parsing.lexer.tokens module

```

class robot.parsing.lexer.tokens.Token(type=None, value=None, lineno=-1, col_offset=-1,
                                         error=None)
    Bases: object

```

Token representing piece of Robot Framework data.

Each token has type, value, line number, column offset and end column offset in `type`, `value`, `lineno`, `col_offset` and `end_col_offset` attributes, respectively. Tokens representing error also have their error message in `error` attribute.

Token types are declared as class attributes such as `SETTING_HEADER` and `EOL`. Values of these constants have changed slightly in Robot Framework 4.0 and they may change again in the future. It is thus safer to use the constants, not their values, when types are needed. For example, use `Token(Token.EOL)` instead of `Token('EOL')` and `token.type == Token.EOL` instead of `token.type == 'EOL'`.

If `value` is not given when `Token` is initialized and `type` is `IF`, `ELSE_IF`, `ELSE`, `FOR`, `END`, `WITH_NAME` or `CONTINUATION`, the value is automatically set to the correct marker value like `'IF'` or `'ELSE IF'`. If `type` is `EOL` in this case, the value is set to `'\n'`.

```

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'
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'
RETURN_SETTING = 'RETURN'
NAME = 'NAME'
VARIABLE = 'VARIABLE'
ARGUMENT = 'ARGUMENT'
ASSIGN = 'ASSIGN'
KEYWORD = 'KEYWORD'
WITH_NAME = 'WITH NAME'
FOR = 'FOR'
FOR_SEPARATOR = 'FOR SEPARATOR'
END = 'END'
IF = 'IF'
INLINE_IF = 'INLINE IF'
ELSE_IF = 'ELSE IF'
ELSE = 'ELSE'
TRY = 'TRY'
EXCEPT = 'EXCEPT'
FINALLY = 'FINALLY'
AS = 'AS'
WHILE = 'WHILE'
```



```

RETURN_STATEMENT = 'RETURN STATEMENT'
CONTINUE = 'CONTINUE'
BREAK = 'BREAK'
OPTION = 'OPTION'
SEPARATOR = 'SEPARATOR'
COMMENT = 'COMMENT'
CONTINUATION = 'CONTINUATION'
EOL = 'EOL'
EOS = 'EOS'
ERROR = 'ERROR'
FATAL_ERROR = 'FATAL ERROR'
NON_DATA_TOKENS = frozenset({'EOL', 'COMMENT', 'SEPARATOR', 'CONTINUATION', 'EOS'})
SETTING_TOKENS = frozenset({'RETURN', 'SUITE TEARDOWN', 'TEST SETUP', 'TAGS', 'TEMPLATE'})
HEADER_TOKENS = frozenset({'COMMENT HEADER', 'KEYWORD HEADER', 'SETTING HEADER', 'VARIABLE'})
ALLOW_VARIABLES = frozenset({'TESTCASE NAME', 'KEYWORD NAME', 'NAME', 'ARGUMENT'})

type
value
lineno
col_offset
error
end_col_offset
set_error(error, fatal=False)
tokenize_variables()
    Tokenizes possible variables in token value.

    Yields the token itself if the token does not allow variables (see Token.ALLOW_VARIABLES) or its
    value does not contain variables. Otherwise yields variable tokens as well as tokens before, after, or
    between variables so that they have the same type as the original token.

class robot.parsing.lexer.tokens.EOS(lineno=-1, col_offset=-1)
    Bases: robot.parsing.lexer.tokens.Token
    Token representing end of a statement.

    classmethod from_token(token, before=False)

    ALLOW_VARIABLES = frozenset({'TESTCASE NAME', 'KEYWORD NAME', 'NAME', 'ARGUMENT'})
    ARGUMENT = 'ARGUMENT'
    ARGUMENTS = 'ARGUMENTS'
    AS = 'AS'
    ASSIGN = 'ASSIGN'
    BREAK = 'BREAK'

```

```
COMMENT = 'COMMENT'
COMMENT_HEADER = 'COMMENT HEADER'
CONTINUATION = 'CONTINUATION'
CONTINUE = 'CONTINUE'
DEFAULT_TAGS = 'DEFAULT TAGS'
DOCUMENTATION = 'DOCUMENTATION'
ELSE = 'ELSE'
ELSE_IF = 'ELSE IF'
END = 'END'
EOL = 'EOL'
EOS = 'EOS'
ERROR = 'ERROR'
EXCEPT = 'EXCEPT'
FATAL_ERROR = 'FATAL ERROR'
FINALLY = 'FINALLY'
FOR = 'FOR'
FORCE_TAGS = 'FORCE TAGS'
FOR_SEPARATOR = 'FOR SEPARATOR'
HEADER_TOKENS = frozenset({'COMMENT HEADER', 'KEYWORD HEADER', 'SETTING HEADER', 'VARI
IF = 'IF'
INLINE_IF = 'INLINE IF'
KEYWORD = 'KEYWORD'
KEYWORD_HEADER = 'KEYWORD HEADER'
KEYWORD_NAME = 'KEYWORD NAME'
LIBRARY = 'LIBRARY'
METADATA = 'METADATA'
NAME = 'NAME'
NON_DATA_TOKENS = frozenset({'EOL', 'COMMENT', 'SEPARATOR', 'CONTINUATION', 'EOS'})
OPTION = 'OPTION'
RESOURCE = 'RESOURCE'
RETURN = 'RETURN'
RETURN_SETTING = 'RETURN'
RETURN_STATEMENT = 'RETURN STATEMENT'
SEPARATOR = 'SEPARATOR'
SETTING_HEADER = 'SETTING HEADER'
SETTING_TOKENS = frozenset({'RETURN', 'SUITE TEARDOWN', 'TEST SETUP', 'TAGS', 'TEMPLAT
```

```

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'
TRY = 'TRY'
VARIABLE = 'VARIABLE'
VARIABLES = 'VARIABLES'
VARIABLE_HEADER = 'VARIABLE HEADER'
WHILE = 'WHILE'
WITH_NAME = 'WITH NAME'

col_offset
end_col_offset

error

lineno

set_error(error, fatal=False)

tokenize_variables()
    Tokenizes possible variables in token value.

    Yields the token itself if the token does not allow variables (see Token.ALLOW_VARIABLES) or its
    value does not contain variables. Otherwise yields variable tokens as well as tokens before, after, or
    between variables so that they have the same type as the original token.

type
value

class robot.parsing.lexer.tokens.END(lineno=-1, col_offset=-1, virtual=False)
    Bases: robot.parsing.lexer.tokens.Token

    Token representing END token used to signify block ending.

    Virtual END tokens have “” as their value, with “real” END tokens the value is ‘END’.

    classmethod from_token(token, virtual=False)

ALLOW_VARIABLES = frozenset({'TESTCASE NAME', 'KEYWORD NAME', 'NAME', 'ARGUMENT'})

```

```
ARGUMENT = 'ARGUMENT'
ARGUMENTS = 'ARGUMENTS'
AS = 'AS'
ASSIGN = 'ASSIGN'
BREAK = 'BREAK'
COMMENT = 'COMMENT'
COMMENT_HEADER = 'COMMENT HEADER'
CONTINUATION = 'CONTINUATION'
CONTINUE = 'CONTINUE'
DEFAULT_TAGS = 'DEFAULT TAGS'
DOCUMENTATION = 'DOCUMENTATION'
ELSE = 'ELSE'
ELSE_IF = 'ELSE IF'
END = 'END'
EOL = 'EOL'
EOS = 'EOS'
ERROR = 'ERROR'
EXCEPT = 'EXCEPT'
FATAL_ERROR = 'FATAL ERROR'
FINALLY = 'FINALLY'
FOR = 'FOR'
FORCE_TAGS = 'FORCE TAGS'
FOR_SEPARATOR = 'FOR SEPARATOR'
HEADER_TOKENS = frozenset({'COMMENT HEADER', 'KEYWORD HEADER', 'SETTING HEADER', 'VARIABLE HEADER'})
IF = 'IF'
INLINE_IF = 'INLINE IF'
KEYWORD = 'KEYWORD'
KEYWORD_HEADER = 'KEYWORD HEADER'
KEYWORD_NAME = 'KEYWORD NAME'
LIBRARY = 'LIBRARY'
METADATA = 'METADATA'
NAME = 'NAME'
NON_DATA_TOKENS = frozenset({'EOL', 'COMMENT', 'SEPARATOR', 'CONTINUATION', 'EOS'})
OPTION = 'OPTION'
RESOURCE = 'RESOURCE'
RETURN = 'RETURN'
```

```

RETURN_SETTING = 'RETURN'
RETURN_STATEMENT = 'RETURN STATEMENT'
SEPARATOR = 'SEPARATOR'
SETTING_HEADER = 'SETTING HEADER'
SETTING_TOKENS = frozenset({'RETURN', 'SUITE TEARDOWN', 'TEST SETUP', 'TAGS', 'TEMPLAT
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'
TRY = 'TRY'
VARIABLE = 'VARIABLE'
VARIABLES = 'VARIABLES'
VARIABLE_HEADER = 'VARIABLE HEADER'
WHILE = 'WHILE'
WITH_NAME = 'WITH NAME'
col_offset
end_col_offset
error
lineno
set_error(error, fatal=False)
tokenize_variables()
    Tokenizes possible variables in token value.

    Yields the token itself if the token does not allow variables (see Token.ALLOW_VARIABLES) or its
    value does not contain variables. Otherwise yields variable tokens as well as tokens before, after, or
    between variables so that they have the same type as the original token.

type
value

```

**robot.parsing.model package****Submodules****robot.parsing.model.blocks module**

```
class robot.parsing.model.blocks.Block
    Bases: _ast.AST

    errors = ()

    lineno

    col_offset

    end_lineno

    end_col_offset

    validate_model()

    validate(context)

class robot.parsing.model.blocks.HeaderAndBody(header, body=None, errors=())
    Bases: robot.parsing.model.blocks.Block

    errors = ()

    col_offset

    end_col_offset

    end_lineno

    lineno

    validate(context)

    validate_model()

class robot.parsing.model.blocks.File(sections=None, source=None)
    Bases: robot.parsing.model.blocks.Block

    save(output=None)
        Save model to the given output or to the original source file.

        The output can be a path to a file or an already opened file object. If output is not given, the original
        source file will be overwritten.

    col_offset

    end_col_offset

    end_lineno

    errors = ()

    lineno

    validate(context)

    validate_model()

class robot.parsing.model.blocks.Section(header=None, body=None)
    Bases: robot.parsing.model.blocks.Block
```

```
col_offset
end_col_offset
end_lineno
errors = ()
lineno
validate(context)
validate_model()

class robot.parsing.model.blocks.SettingSection(header=None, body=None)
    Bases: robot.parsing.model.blocks.Section
    col_offset
    end_col_offset
    end_lineno
    errors = ()
    lineno
    validate(context)
    validate_model()

class robot.parsing.model.blocks.VariableSection(header=None, body=None)
    Bases: robot.parsing.model.blocks.Section
    col_offset
    end_col_offset
    end_lineno
    errors = ()
    lineno
    validate(context)
    validate_model()

class robot.parsing.model.blocks.TestCaseSection(header=None, body=None)
    Bases: robot.parsing.model.blocks.Section
    tasks
    col_offset
    end_col_offset
    end_lineno
    errors = ()
    lineno
    validate(context)
    validate_model()

class robot.parsing.model.blocks.KeywordSection(header=None, body=None)
    Bases: robot.parsing.model.blocks.Section
```

```
col_offset
end_col_offset
end_lineno
errors = ()
lineno
validate(context)
validate_model()

class robot.parsing.model.blocks.CommentSection(header=None, body=None)
    Bases: robot.parsing.model.blocks.Section
    col_offset
    end_col_offset
    end_lineno
    errors = ()
    lineno
    validate(context)
    validate_model()

class robot.parsing.model.blocks.TestCase(header, body=None)
    Bases: robot.parsing.model.blocks.Block
    name
    col_offset
    end_col_offset
    end_lineno
    errors = ()
    lineno
    validate(context)
    validate_model()

class robot.parsing.model.blocks.Keyword(header, body=None)
    Bases: robot.parsing.model.blocks.Block
    name
    col_offset
    end_col_offset
    end_lineno
    errors = ()
    lineno
    validate(context)
    validate_model()
```



```
class robot.parsing.model.blocks.If(header, body=None, or_else=None, end=None, errors=())
```

Bases: [robot.parsing.model.blocks.Block](#)

Represents IF structures in the model.

Used with IF, Inline IF, ELSE IF and ELSE nodes. The *type* attribute specifies the type.

**errors** = ()

**type**

**condition**

**assign**

**validate**(context)

**col\_offset**

**end\_col\_offset**

**end\_lineno**

**lineno**

**validate\_model**()

```
class robot.parsing.model.blocks.For(header, body=None, end=None, errors=())
```

Bases: [robot.parsing.model.blocks.Block](#)

**errors** = ()

**variables**

**values**

**flavor**

**validate**(context)

**col\_offset**

**end\_col\_offset**

**end\_lineno**

**lineno**

**validate\_model**()

```
class robot.parsing.model.blocks.Try(header, body=None, next=None, end=None, errors=())
```

Bases: [robot.parsing.model.blocks.Block](#)

**errors** = ()

**type**

**patterns**

**pattern\_type**

**variable**

**validate**(context)

**col\_offset**

**end\_col\_offset**

```
    end_lineno
    lineno
    validate_model()

class robot.parsing.model.blocks.While(header, body=None, end=None, errors=())
    Bases: robot.parsing.model.blocks.Block
    errors = ()
    condition
    limit
    validate(context)
    col_offset
    end_col_offset
    end_lineno
    lineno
    validate_model()

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.ModelValidator
    Bases: robot.parsing.model.visitor.ModelVisitor
    visit_Block(node)
    visit_Try(node)
    visit_Statement(node)
    generic_visit(node)
        Called if no explicit visitor function exists for a node.
    visit(node)
        Visit a node.

class robot.parsing.model.blocks.ValidationContext
    Bases: object
    start_block(node)
    end_block()
    in_keyword
    in_for
    in_while
```

```
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, errors=())
```

```
    Bases: _ast.AST
```

```
    type = None
```

```
    handles_types = (())
```

```
    lineno
```

```
    col_offset
```

```
    end_lineno
```

```
    end_col_offset
```

```
    classmethod register(subcls)
```

```
    classmethod from_tokens(tokens)
```

```
    classmethod from_params(*args, **kwargs)
```

Create statement from passed parameters.

Required and optional arguments should match class properties. Values are used to create matching tokens.

There is one notable difference for *Documentation* statement where `settings_header` flag is used to determine if statement belongs to settings header or test/keyword.

Most implementations support following general properties: - *separator* whitespace inserted between each token. Default is four spaces. - *indent* whitespace inserted before first token. Default is four spaces. - *eol* end of line sign. Default is '\n'.

```
    data_tokens
```

```
    get_token(*types)
```

Return a token with the given type.

If there are no matches, return None. If there are multiple matches, return the first match.

**get\_tokens** (\*types)  
Return tokens having any of the given types.

**get\_value** (type, default=None)  
Return value of a token with the given type.  
If there are no matches, return default. If there are multiple matches, return the value of the first match.

**get\_values** (\*types)  
Return values of tokens having any of the given types.

**lines**

**validate** (context)

**class** robot.parsing.model.statements.**DocumentationOrMetadata** (tokens, errors=())  
Bases: *robot.parsing.model.statements.Statement*

**col\_offset**

**data\_tokens**

**end\_col\_offset**

**end\_lineno**

**classmethod from\_params** (\*args, \*\*kwargs)  
Create statement from passed parameters.  
Required and optional arguments should match class properties. Values are used to create matching tokens.  
There is one notable difference for *Documentation* statement where *settings\_header* flag is used to determine if statement belongs to settings header or test/keyword.  
Most implementations support following general properties: - *separator* whitespace inserted between each token. Default is four spaces. - *indent* whitespace inserted before first token. Default is four spaces. - *eol* end of line sign. Default is '\n'.

**classmethod from\_tokens** (tokens)

**get\_token** (\*types)  
Return a token with the given type.  
If there are no matches, return None. If there are multiple matches, return the first match.

**get\_tokens** (\*types)  
Return tokens having any of the given types.

**get\_value** (type, default=None)  
Return value of a token with the given type.  
If there are no matches, return default. If there are multiple matches, return the value of the first match.

**get\_values** (\*types)  
Return values of tokens having any of the given types.

**handles\_types** = ()

**lineno**

**lines**

**classmethod register** (subcls)

**type** = None

**validate** (context)

```

class robot.parsing.model.statements.SingleValue(tokens, errors=())
    Bases: robot.parsing.model.statements.Statement

    value
    col_offset
    data_tokens
    end_col_offset
    end_lineno

    classmethod from_params(*args, **kwargs)
        Create statement from passed parameters.

        Required and optional arguments should match class properties. Values are used to create matching tokens.

        There is one notable difference for Documentation statement where settings_header flag is used to
        determine if statement belongs to settings header or test/keyword.

        Most implementations support following general properties: - separator whitespace inserted between each
        token. Default is four spaces. - indent whitespace inserted before first token. Default is four spaces. -
        eol end of line sign. Default is '\n'.

    classmethod from_tokens(tokens)

    get_token(*types)
        Return a token with the given type.

        If there are no matches, return None. If there are multiple matches, return the first match.

    get_tokens(*types)
        Return tokens having any of the given types.

    get_value(type, default=None)
        Return value of a token with the given type.

        If there are no matches, return default. If there are multiple matches, return the value of the first match.

    get_values(*types)
        Return values of tokens having any of the given types.

    handles_types = ()

    lineno
    lines

    classmethod register(subcls)

    type = None

    validate(context)

class robot.parsing.model.statements.MultiValue(tokens, errors=())
    Bases: robot.parsing.model.statements.Statement

    values
    col_offset
    data_tokens
    end_col_offset
    end_lineno

```

**classmethod from\_params** (\*args, \*\*kwargs)

Create statement from passed parameters.

Required and optional arguments should match class properties. Values are used to create matching tokens.

There is one notable difference for *Documentation* statement where `settings_header` flag is used to determine if statement belongs to settings header or test/keyword.

Most implementations support following general properties: - *separator* whitespace inserted between each token. Default is four spaces. - *indent* whitespace inserted before first token. Default is four spaces. - *eol* end of line sign. Default is `'\n'`.

**classmethod from\_tokens** (tokens)

**get\_token** (\*types)

Return a token with the given type.

If there are no matches, return `None`. If there are multiple matches, return the first match.

**get\_tokens** (\*types)

Return tokens having any of the given types.

**get\_value** (type, default=None)

Return value of a token with the given type.

If there are no matches, return `default`. If there are multiple matches, return the value of the first match.

**get\_values** (\*types)

Return values of tokens having any of the given types.

**handles\_types** = ()

**lineno**

**lines**

**classmethod register** (subcls)

**type** = None

**validate** (context)

**class** robot.parsing.model.statements.**Fixture** (tokens, errors=())

Bases: `robot.parsing.model.statements.Statement`

**name**

**args**

**col\_offset**

**data\_tokens**

**end\_col\_offset**

**end\_lineno**

**classmethod from\_params** (\*args, \*\*kwargs)

Create statement from passed parameters.

Required and optional arguments should match class properties. Values are used to create matching tokens.

There is one notable difference for *Documentation* statement where `settings_header` flag is used to determine if statement belongs to settings header or test/keyword.

Most implementations support following general properties: - *separator* whitespace inserted between each token. Default is four spaces. - *indent* whitespace inserted before first token. Default is four spaces. - *eol* end of line sign. Default is '`\n`'.

**classmethod** `from_tokens` (*tokens*)

**get\_token** (*\*types*)

Return a token with the given *type*.

If there are no matches, return `None`. If there are multiple matches, return the first match.

**get\_tokens** (*\*types*)

Return tokens having any of the given *types*.

**get\_value** (*type, default=None*)

Return value of a token with the given *type*.

If there are no matches, return *default*. If there are multiple matches, return the value of the first match.

**get\_values** (*\*types*)

Return values of tokens having any of the given *types*.

**handles\_types** = ()

**lineno**

**lines**

**classmethod** `register` (*subcls*)

**type** = `None`

**validate** (*context*)

**class** `robot.parsing.model.statements.SectionHeader` (*tokens, errors=()*)

Bases: `robot.parsing.model.statements.Statement`

**handles\_types** = ('SETTING HEADER', 'VARIABLE HEADER', 'TESTCASE HEADER', 'KEYWORD HEAD

**classmethod** `from_params` (*type, name=None, eol='\n'*)

Create statement from passed parameters.

Required and optional arguments should match class properties. Values are used to create matching tokens.

There is one notable difference for *Documentation* statement where `settings_header` flag is used to determine if statement belongs to settings header or test/keyword.

Most implementations support following general properties: - *separator* whitespace inserted between each token. Default is four spaces. - *indent* whitespace inserted before first token. Default is four spaces. - *eol* end of line sign. Default is '`\n`'.

**type**

**name**

**col\_offset**

**data\_tokens**

**end\_col\_offset**

**end\_lineno**

**classmethod** `from_tokens` (*tokens*)

**get\_token** (\*types)

Return a token with the given type.

If there are no matches, return None. If there are multiple matches, return the first match.

**get\_tokens** (\*types)

Return tokens having any of the given types.

**get\_value** (type, default=None)

Return value of a token with the given type.

If there are no matches, return default. If there are multiple matches, return the value of the first match.

**get\_values** (\*types)

Return values of tokens having any of the given types.

**lineno**

**lines**

**classmethod register** (subcls)

**validate** (context)

**class** robot.parsing.model.statements.**LibraryImport** (tokens, errors=())

Bases: *robot.parsing.model.statements.Statement*

**type** = 'LIBRARY'

**classmethod from\_params** (name, args=(), alias=None, separator=' ', eol='\n')

Create statement from passed parameters.

Required and optional arguments should match class properties. Values are used to create matching tokens.

There is one notable difference for *Documentation* statement where `settings_header` flag is used to determine if statement belongs to settings header or test/keyword.

Most implementations support following general properties: - *separator* whitespace inserted between each token. Default is four spaces. - *indent* whitespace inserted before first token. Default is four spaces. - *eol* end of line sign. Default is '\n'.

**name**

**args**

**alias**

**col\_offset**

**data\_tokens**

**end\_col\_offset**

**end\_lineno**

**classmethod from\_tokens** (tokens)

**get\_token** (\*types)

Return a token with the given type.

If there are no matches, return None. If there are multiple matches, return the first match.

**get\_tokens** (\*types)

Return tokens having any of the given types.



```

get_value (type, default=None)
    Return value of a token with the given type.

    If there are no matches, return default. If there are multiple matches, return the value of the first match.

get_values (*types)
    Return values of tokens having any of the given types.

handles_types = ()

lineno

lines

classmethod register (subcls)

validate (context)

class robot.parsing.model.statements.ResourceImport (tokens, errors=())
    Bases: robot.parsing.model.statements.Statement

    type = 'RESOURCE'

    classmethod from_params (name, separator=' ', eol='\n')
        Create statement from passed parameters.

        Required and optional arguments should match class properties. Values are used to create matching tokens.

        There is one notable difference for Documentation statement where settings_header flag is used to
        determine if statement belongs to settings header or test/keyword.

        Most implementations support following general properties: - separator whitespace inserted between each
        token. Default is four spaces. - indent whitespace inserted before first token. Default is four spaces. -
        eol end of line sign. Default is '\n'.

    name

    col_offset

    data_tokens

    end_col_offset

    end_lineno

    classmethod from_tokens (tokens)

    get_token (*types)
        Return a token with the given type.

        If there are no matches, return None. If there are multiple matches, return the first match.

    get_tokens (*types)
        Return tokens having any of the given types.

    get_value (type, default=None)
        Return value of a token with the given type.

        If there are no matches, return default. If there are multiple matches, return the value of the first match.

    get_values (*types)
        Return values of tokens having any of the given types.

    handles_types = ()

    lineno

    lines

```

```
classmethod register(subcls)

validate(context)

class robot.parsing.model.statements.VariablesImport(tokens, errors=())
    Bases: robot.parsing.model.statements.Statement

    type = 'VARIABLES'

    classmethod from_params(name, args=(), separator=' ', eol='\n')
        Create statement from passed parameters.

        Required and optional arguments should match class properties. Values are used to create matching tokens.

        There is one notable difference for Documentation statement where settings_header flag is used to
        determine if statement belongs to settings header or test/keyword.

        Most implementations support following general properties: - separator whitespace inserted between each
        token. Default is four spaces. - indent whitespace inserted before first token. Default is four spaces. -
        eol end of line sign. Default is '\n'.

    name
    args
    col_offset
    data_tokens
    end_col_offset
    end_lineno

    classmethod from_tokens(tokens)

    get_token(*types)
        Return a token with the given type.

        If there are no matches, return None. If there are multiple matches, return the first match.

    get_tokens(*types)
        Return tokens having any of the given types.

    get_value(type, default=None)
        Return value of a token with the given type.

        If there are no matches, return default. If there are multiple matches, return the value of the first match.

    get_values(*types)
        Return values of tokens having any of the given types.

    handles_types = ()

    lineno
    lines

    classmethod register(subcls)

    validate(context)

class robot.parsing.model.statements.Documentation(tokens, errors=())
    Bases: robot.parsing.model.statements.DocumentationOrMetadata

    type = 'DOCUMENTATION'
```

**classmethod from\_params** (*value*, *indent*=' ', *separator*=' ', *eol*='\n', *settings\_section*=True)

Create statement from passed parameters.

Required and optional arguments should match class properties. Values are used to create matching tokens.

There is one notable difference for *Documentation* statement where *settings\_header* flag is used to determine if statement belongs to settings header or test/keyword.

Most implementations support following general properties: - *separator* whitespace inserted between each token. Default is four spaces. - *indent* whitespace inserted before first token. Default is four spaces. - *eol* end of line sign. Default is '\n'.

**value**

**col\_offset**

**data\_tokens**

**end\_col\_offset**

**end\_lineno**

**classmethod from\_tokens** (*tokens*)

**get\_token** (*\*types*)

Return a token with the given type.

If there are no matches, return None. If there are multiple matches, return the first match.

**get\_tokens** (*\*types*)

Return tokens having any of the given types.

**get\_value** (*type*, *default*=None)

Return value of a token with the given type.

If there are no matches, return default. If there are multiple matches, return the value of the first match.

**get\_values** (*\*types*)

Return values of tokens having any of the given types.

**handles\_types** = ()

**lineno**

**lines**

**classmethod register** (*subcls*)

**validate** (*context*)

**class** robot.parsing.model.statements.**Metadata** (*tokens*, *errors*=())

Bases: *robot.parsing.model.statements.DocumentationOrMetadata*

**type** = 'METADATA'

**classmethod from\_params** (*name*, *value*, *separator*=' ', *eol*='\n')

Create statement from passed parameters.

Required and optional arguments should match class properties. Values are used to create matching tokens.

There is one notable difference for *Documentation* statement where *settings\_header* flag is used to determine if statement belongs to settings header or test/keyword.

Most implementations support following general properties: - *separator* whitespace inserted between each token. Default is four spaces. - *indent* whitespace inserted before first token. Default is four spaces. - *eol* end of line sign. Default is '\n'.

```
name
value
col_offset
data_tokens
end_col_offset
end_lineno
classmethod from_tokens (tokens)
get_token (*types)
    Return a token with the given type.

    If there are no matches, return None. If there are multiple matches, return the first match.
get_tokens (*types)
    Return tokens having any of the given types.
get_value (type, default=None)
    Return value of a token with the given type.

    If there are no matches, return default. If there are multiple matches, return the value of the first match.
get_values (*types)
    Return values of tokens having any of the given types.
handles_types = ()
lineno
lines
classmethod register (subcls)
validate (context)
class robot.parsing.model.statements.ForceTags (tokens, errors=())
    Bases: robot.parsing.model.statements.MultiValue
    type = 'FORCE TAGS'
    classmethod from_params (values, separator=' ', eol='\n')
        Create statement from passed parameters.

        Required and optional arguments should match class properties. Values are used to create matching tokens.

        There is one notable difference for Documentation statement where settings_header flag is used to
        determine if statement belongs to settings header or test/keyword.

        Most implementations support following general properties: - separator whitespace inserted between each
        token. Default is four spaces. - indent whitespace inserted before first token. Default is four spaces. -
        eol end of line sign. Default is '\n'.

    col_offset
    data_tokens
    end_col_offset
    end_lineno
    classmethod from_tokens (tokens)
```

```

get_token (*types)
    Return a token with the given type.

    If there are no matches, return None. If there are multiple matches, return the first match.

get_tokens (*types)
    Return tokens having any of the given types.

get_value (type, default=None)
    Return value of a token with the given type.

    If there are no matches, return default. If there are multiple matches, return the value of the first match.

get_values (*types)
    Return values of tokens having any of the given types.

handles_types = ()

lineno

lines

classmethod register (subcls)

validate (context)

values

class robot.parsing.model.statements.DefaultTags (tokens, errors=())
    Bases: robot.parsing.model.statements.MultiValue

    type = 'DEFAULT TAGS'

    classmethod from_params (values, separator=' ', eol='\n')
        Create statement from passed parameters.

        Required and optional arguments should match class properties. Values are used to create matching tokens.

        There is one notable difference for Documentation statement where settings_header flag is used to
        determine if statement belongs to settings header or test/keyword.

        Most implementations support following general properties: - separator whitespace inserted between each
        token. Default is four spaces. - indent whitespace inserted before first token. Default is four spaces. -
        eol end of line sign. Default is '\n'.

    col_offset

    data_tokens

    end_col_offset

    end_lineno

    classmethod from_tokens (tokens)

    get_token (*types)
        Return a token with the given type.

        If there are no matches, return None. If there are multiple matches, return the first match.

    get_tokens (*types)
        Return tokens having any of the given types.

    get_value (type, default=None)
        Return value of a token with the given type.

        If there are no matches, return default. If there are multiple matches, return the value of the first match.

```

```
get_values (*types)
    Return values of tokens having any of the given types.

handles_types = ()

lineno

lines

classmethod register (subcls)

validate (context)

values
```

**class** robot.parsing.model.statements.**SuiteSetup** (tokens, errors=())  
Bases: *robot.parsing.model.statements.Fixture*

**type** = 'SUITE SETUP'

**classmethod from\_params** (name, args=(), separator=' ', eol='\n')  
Create statement from passed parameters.

Required and optional arguments should match class properties. Values are used to create matching tokens.

There is one notable difference for *Documentation* statement where `settings_header` flag is used to determine if statement belongs to settings header or test/keyword.

Most implementations support following general properties: - *separator* whitespace inserted between each token. Default is four spaces. - *indent* whitespace inserted before first token. Default is four spaces. - *eol* end of line sign. Default is '\n'.

**args**

**col\_offset**

**data\_tokens**

**end\_col\_offset**

**end\_lineno**

**classmethod from\_tokens** (tokens)

**get\_token** (\*types)  
Return a token with the given type.

If there are no matches, return None. If there are multiple matches, return the first match.

**get\_tokens** (\*types)  
Return tokens having any of the given types.

**get\_value** (type, default=None)  
Return value of a token with the given type.

If there are no matches, return default. If there are multiple matches, return the value of the first match.

**get\_values** (\*types)  
Return values of tokens having any of the given types.

**handles\_types** = ()

**lineno**

**lines**

**name**

```

    classmethod register (subcls)
    validate (context)
class robot.parsing.model.statements.SuiteTeardown (tokens, errors=())
    Bases: robot.parsing.model.statements.Fixture
    type = 'SUITE TEARDOWN'
    classmethod from_params (name, args=(), separator=' ', eol='\n')
        Create statement from passed parameters.

        Required and optional arguments should match class properties. Values are used to create matching tokens.

        There is one notable difference for Documentation statement where settings_header flag is used to
        determine if statement belongs to settings header or test/keyword.

        Most implementations support following general properties: - separator whitespace inserted between each
        token. Default is four spaces. - indent whitespace inserted before first token. Default is four spaces. -
        eol end of line sign. Default is '\n'.

    args
    col_offset
    data_tokens
    end_col_offset
    end_lineno
    classmethod from_tokens (tokens)
    get_token (*types)
        Return a token with the given type.

        If there are no matches, return None. If there are multiple matches, return the first match.
    get_tokens (*types)
        Return tokens having any of the given types.
    get_value (type, default=None)
        Return value of a token with the given type.

        If there are no matches, return default. If there are multiple matches, return the value of the first match.
    get_values (*types)
        Return values of tokens having any of the given types.
    handles_types = ()
    lineno
    lines
    name
    classmethod register (subcls)
    validate (context)
class robot.parsing.model.statements.TestSetup (tokens, errors=())
    Bases: robot.parsing.model.statements.Fixture
    type = 'TEST SETUP'

```

```
classmethod from_params (name, args=(), separator=' ', eol='\n')
```

Create statement from passed parameters.

Required and optional arguments should match class properties. Values are used to create matching tokens.

There is one notable difference for *Documentation* statement where `settings_header` flag is used to determine if statement belongs to settings header or test/keyword.

Most implementations support following general properties: - *separator* whitespace inserted between each token. Default is four spaces. - *indent* whitespace inserted before first token. Default is four spaces. - *eol* end of line sign. Default is `'\n'`.

**args**

**col\_offset**

**data\_tokens**

**end\_col\_offset**

**end\_lineno**

```
classmethod from_tokens (tokens)
```

```
get_token (*types)
```

Return a token with the given type.

If there are no matches, return `None`. If there are multiple matches, return the first match.

```
get_tokens (*types)
```

Return tokens having any of the given types.

```
get_value (type, default=None)
```

Return value of a token with the given type.

If there are no matches, return `default`. If there are multiple matches, return the value of the first match.

```
get_values (*types)
```

Return values of tokens having any of the given types.

```
handles_types = ()
```

**lineno**

**lines**

**name**

```
classmethod register (subcls)
```

```
validate (context)
```

```
class robot.parsing.model.statements.TestTeardown (tokens, errors=())
```

Bases: `robot.parsing.model.statements.Fixture`

```
type = 'TEST TEARDOWN'
```

```
classmethod from_params (name, args=(), separator=' ', eol='\n')
```

Create statement from passed parameters.

Required and optional arguments should match class properties. Values are used to create matching tokens.

There is one notable difference for *Documentation* statement where `settings_header` flag is used to determine if statement belongs to settings header or test/keyword.



Most implementations support following general properties: - *separator* whitespace inserted between each token. Default is four spaces. - *indent* whitespace inserted before first token. Default is four spaces. - *eol* end of line sign. Default is '`\n`'.

```

args
col_offset
data_tokens
end_col_offset
end_lineno
classmethod from_tokens (tokens)
get_token (*types)
    Return a token with the given type.

    If there are no matches, return None. If there are multiple matches, return the first match.
get_tokens (*types)
    Return tokens having any of the given types.
get_value (type, default=None)
    Return value of a token with the given type.

    If there are no matches, return default. If there are multiple matches, return the value of the first match.
get_values (*types)
    Return values of tokens having any of the given types.
handles_types = ()
lineno
lines
name
classmethod register (subcls)
validate (context)
class robot.parsing.model.statements.TestTemplate (tokens, errors=())
    Bases: robot.parsing.model.statements.SingleValue
    type = 'TEST TEMPLATE'
    classmethod from_params (value, separator=' ', eol='\n')
        Create statement from passed parameters.

        Required and optional arguments should match class properties. Values are used to create matching tokens.

        There is one notable difference for Documentation statement where settings_header flag is used to
        determine if statement belongs to settings header or test/keyword.

        Most implementations support following general properties: - separator whitespace inserted between each
        token. Default is four spaces. - indent whitespace inserted before first token. Default is four spaces. -
        eol end of line sign. Default is '\n'.
col_offset
data_tokens
end_col_offset
end_lineno

```

```
classmethod from_tokens (tokens)

get_token (*types)
    Return a token with the given type.

    If there are no matches, return None. If there are multiple matches, return the first match.

get_tokens (*types)
    Return tokens having any of the given types.

get_value (type, default=None)
    Return value of a token with the given type.

    If there are no matches, return default. If there are multiple matches, return the value of the first match.

get_values (*types)
    Return values of tokens having any of the given types.

handles_types = ()

lineno

lines

classmethod register (subcls)

validate (context)

value

class robot.parsing.model.statements.TestTimeout (tokens, errors=())
    Bases: robot.parsing.model.statements.SingleValue

    type = 'TEST TIMEOUT'

    classmethod from_params (value, separator=' ', eol='\n')
        Create statement from passed parameters.

        Required and optional arguments should match class properties. Values are used to create matching tokens.

        There is one notable difference for Documentation statement where settings_header flag is used to
        determine if statement belongs to settings header or test/keyword.

        Most implementations support following general properties: - separator whitespace inserted between each
        token. Default is four spaces. - indent whitespace inserted before first token. Default is four spaces. -
        eol end of line sign. Default is '\n'.

    col_offset

    data_tokens

    end_col_offset

    end_lineno

    classmethod from_tokens (tokens)

    get_token (*types)
        Return a token with the given type.

        If there are no matches, return None. If there are multiple matches, return the first match.

    get_tokens (*types)
        Return tokens having any of the given types.
```

```

get_value (type, default=None)
    Return value of a token with the given type.

    If there are no matches, return default. If there are multiple matches, return the value of the first match.

get_values (*types)
    Return values of tokens having any of the given types.

handles_types = ()

lineno

lines

classmethod register (subcls)

validate (context)

value

class robot.parsing.model.statements.Variable (tokens, errors=())
    Bases: robot.parsing.model.statements.Statement

    type = 'VARIABLE'

    classmethod from_params (name, value, separator=' ', eol='\n')
        Create statement from passed parameters.

        Required and optional arguments should match class properties. Values are used to create matching tokens.

        There is one notable difference for Documentation statement where settings_header flag is used to
        determine if statement belongs to settings header or test/keyword.

        Most implementations support following general properties: - separator whitespace inserted between each
        token. Default is four spaces. - indent whitespace inserted before first token. Default is four spaces. -
        eol end of line sign. Default is '\n'.

    name

    value

    validate (context)

    col_offset

    data_tokens

    end_col_offset

    end_lineno

    classmethod from_tokens (tokens)

    get_token (*types)
        Return a token with the given type.

        If there are no matches, return None. If there are multiple matches, return the first match.

    get_tokens (*types)
        Return tokens having any of the given types.

    get_value (type, default=None)
        Return value of a token with the given type.

        If there are no matches, return default. If there are multiple matches, return the value of the first match.

    get_values (*types)
        Return values of tokens having any of the given types.

```

```
handles_types = ()
lineno
lines
classmethod register(subcls)
```

**class** robot.parsing.model.statements.**TestCaseName**(tokens, errors=())  
Bases: *robot.parsing.model.statements.Statement*

**type** = 'TESTCASE NAME'

**classmethod from\_params**(name, eol='\n')  
Create statement from passed parameters.

Required and optional arguments should match class properties. Values are used to create matching tokens.

There is one notable difference for *Documentation* statement where `settings_header` flag is used to determine if statement belongs to settings header or test/keyword.

Most implementations support following general properties: - *separator* whitespace inserted between each token. Default is four spaces. - *indent* whitespace inserted before first token. Default is four spaces. - *eol* end of line sign. Default is '\n'.

**name**

**col\_offset**

**data\_tokens**

**end\_col\_offset**

**end\_lineno**

**classmethod from\_tokens**(tokens)

**get\_token**(\*types)  
Return a token with the given type.

If there are no matches, return None. If there are multiple matches, return the first match.

**get\_tokens**(\*types)  
Return tokens having any of the given types.

**get\_value**(type, default=None)  
Return value of a token with the given type.

If there are no matches, return default. If there are multiple matches, return the value of the first match.

**get\_values**(\*types)  
Return values of tokens having any of the given types.

```
handles_types = ()
lineno
lines
classmethod register(subcls)
validate(context)
```

**class** robot.parsing.model.statements.**KeywordName**(tokens, errors=())  
Bases: *robot.parsing.model.statements.Statement*

**type** = 'KEYWORD NAME'

**classmethod from\_params** (*name*, *eol*='\\n')

Create statement from passed parameters.

Required and optional arguments should match class properties. Values are used to create matching tokens.

There is one notable difference for *Documentation* statement where `settings_header` flag is used to determine if statement belongs to settings header or test/keyword.

Most implementations support following general properties: - *separator* whitespace inserted between each token. Default is four spaces. - *indent* whitespace inserted before first token. Default is four spaces. - *eol* end of line sign. Default is '\\n'.

**name**

**col\_offset**

**data\_tokens**

**end\_col\_offset**

**end\_lineno**

**classmethod from\_tokens** (*tokens*)

**get\_token** (*\*types*)

Return a token with the given *type*.

If there are no matches, return `None`. If there are multiple matches, return the first match.

**get\_tokens** (*\*types*)

Return tokens having any of the given *types*.

**get\_value** (*type*, *default*=`None`)

Return value of a token with the given *type*.

If there are no matches, return `default`. If there are multiple matches, return the value of the first match.

**get\_values** (*\*types*)

Return values of tokens having any of the given *types*.

**handles\_types** = ()

**lineno**

**lines**

**classmethod register** (*subcls*)

**validate** (*context*)

**class** `robot.parsing.model.statements.Setup` (*tokens*, *errors*=())

Bases: `robot.parsing.model.statements.Fixture`

**type** = 'SETUP'

**classmethod from\_params** (*name*, *args*=(), *indent*=' ', *separator*=' ', *eol*='\\n')

Create statement from passed parameters.

Required and optional arguments should match class properties. Values are used to create matching tokens.

There is one notable difference for *Documentation* statement where `settings_header` flag is used to determine if statement belongs to settings header or test/keyword.

Most implementations support following general properties: - *separator* whitespace inserted between each token. Default is four spaces. - *indent* whitespace inserted before first token. Default is four spaces. - *eol* end of line sign. Default is '\\n'.

```
args
col_offset
data_tokens
end_col_offset
end_lineno

classmethod from_tokens (tokens)

get_token (*types)
    Return a token with the given type.

    If there are no matches, return None. If there are multiple matches, return the first match.

get_tokens (*types)
    Return tokens having any of the given types.

get_value (type, default=None)
    Return value of a token with the given type.

    If there are no matches, return default. If there are multiple matches, return the value of the first match.

get_values (*types)
    Return values of tokens having any of the given types.

handles_types = ()

lineno

lines

name

classmethod register (subcls)

validate (context)

class robot.parsing.model.statements.Teardown (tokens, errors=())
    Bases: robot.parsing.model.statements.Fixture

    type = 'TEARDOWN'

    classmethod from_params (name, args=(), indent=' ', separator=' ', eol='\n')
        Create statement from passed parameters.

        Required and optional arguments should match class properties. Values are used to create matching tokens.

        There is one notable difference for Documentation statement where settings_header flag is used to
        determine if statement belongs to settings header or test/keyword.

        Most implementations support following general properties: - separator whitespace inserted between each
        token. Default is four spaces. - indent whitespace inserted before first token. Default is four spaces. -
        eol end of line sign. Default is '\n'.

args
col_offset
data_tokens
end_col_offset
end_lineno

classmethod from_tokens (tokens)
```

```

get_token (*types)
    Return a token with the given type.

    If there are no matches, return None. If there are multiple matches, return the first match.

get_tokens (*types)
    Return tokens having any of the given types.

get_value (type, default=None)
    Return value of a token with the given type.

    If there are no matches, return default. If there are multiple matches, return the value of the first match.

get_values (*types)
    Return values of tokens having any of the given types.

handles_types = ()

lineno

lines

name

classmethod register (subcls)

validate (context)

class robot.parsing.model.statements.Tags (tokens, errors=())
    Bases: robot.parsing.model.statements.MultiValue

    type = 'TAGS'

    classmethod from_params (values, indent=' ', separator=' ', eol='\n')
        Create statement from passed parameters.

        Required and optional arguments should match class properties. Values are used to create matching tokens.

        There is one notable difference for Documentation statement where settings_header flag is used to
        determine if statement belongs to settings header or test/keyword.

        Most implementations support following general properties: - separator whitespace inserted between each
        token. Default is four spaces. - indent whitespace inserted before first token. Default is four spaces. -
        eol end of line sign. Default is '\n'.

    col_offset

    data_tokens

    end_col_offset

    end_lineno

    classmethod from_tokens (tokens)

    get_token (*types)
        Return a token with the given type.

        If there are no matches, return None. If there are multiple matches, return the first match.

    get_tokens (*types)
        Return tokens having any of the given types.

    get_value (type, default=None)
        Return value of a token with the given type.

        If there are no matches, return default. If there are multiple matches, return the value of the first match.

```

```
get_values (*types)
    Return values of tokens having any of the given types.

handles_types = ()

lineno

lines

classmethod register (subcls)

validate (context)

values
```

**class** robot.parsing.model.statements.Template (tokens, errors=())  
Bases: `robot.parsing.model.statements.SingleValue`

**type** = 'TEMPLATE'

**classmethod** from\_params (value, indent=' ', separator=' ', eol='\n')  
Create statement from passed parameters.

Required and optional arguments should match class properties. Values are used to create matching tokens.

There is one notable difference for *Documentation* statement where `settings_header` flag is used to determine if statement belongs to settings header or test/keyword.

Most implementations support following general properties: - *separator* whitespace inserted between each token. Default is four spaces. - *indent* whitespace inserted before first token. Default is four spaces. - *eol* end of line sign. Default is '\n'.

```
col_offset

data_tokens

end_col_offset

end_lineno

classmethod from_tokens (tokens)

get_token (*types)
    Return a token with the given type.

    If there are no matches, return None. If there are multiple matches, return the first match.

get_tokens (*types)
    Return tokens having any of the given types.

get_value (type, default=None)
    Return value of a token with the given type.

    If there are no matches, return default. If there are multiple matches, return the value of the first match.

get_values (*types)
    Return values of tokens having any of the given types.

handles_types = ()

lineno

lines

classmethod register (subcls)

validate (context)
```



```

    value

class robot.parsing.model.statements.Timeout(tokens, errors=())
    Bases: robot.parsing.model.statements.SingleValue
    type = 'TIMEOUT'
    classmethod from_params(value, indent=' ', separator=' ', eol='\n')
        Create statement from passed parameters.

        Required and optional arguments should match class properties. Values are used to create matching tokens.

        There is one notable difference for Documentation statement where settings_header flag is used to
        determine if statement belongs to settings header or test/keyword.

        Most implementations support following general properties:
        - separator whitespace inserted between each token. Default is four spaces.
        - indent whitespace inserted before first token. Default is four spaces.
        - eol end of line sign. Default is '\n'.

    col_offset
    data_tokens
    end_col_offset
    end_lineno
    classmethod from_tokens(tokens)
    get_token(*types)
        Return a token with the given type.

        If there are no matches, return None. If there are multiple matches, return the first match.

    get_tokens(*types)
        Return tokens having any of the given types.

    get_value(type, default=None)
        Return value of a token with the given type.

        If there are no matches, return default. If there are multiple matches, return the value of the first match.

    get_values(*types)
        Return values of tokens having any of the given types.

    handles_types = ()
    lineno
    lines
    classmethod register(subcls)
    validate(context)
    value

class robot.parsing.model.statements.Arguments(tokens, errors=())
    Bases: robot.parsing.model.statements.MultiValue
    type = 'ARGUMENTS'
    classmethod from_params(args, indent=' ', separator=' ', eol='\n')
        Create statement from passed parameters.

        Required and optional arguments should match class properties. Values are used to create matching tokens.

```

There is one notable difference for *Documentation* statement where `settings_header` flag is used to determine if statement belongs to settings header or test/keyword.

Most implementations support following general properties: - *separator* whitespace inserted between each token. Default is four spaces. - *indent* whitespace inserted before first token. Default is four spaces. - *eol* end of line sign. Default is `'\n'`.

**validate** (*context*)

**col\_offset**

**data\_tokens**

**end\_col\_offset**

**end\_lineno**

**classmethod from\_tokens** (*tokens*)

**get\_token** (*\*types*)

Return a token with the given type.

If there are no matches, return `None`. If there are multiple matches, return the first match.

**get\_tokens** (*\*types*)

Return tokens having any of the given types.

**get\_value** (*type, default=None*)

Return value of a token with the given type.

If there are no matches, return `default`. If there are multiple matches, return the value of the first match.

**get\_values** (*\*types*)

Return values of tokens having any of the given types.

**handles\_types** = ()

**lineno**

**lines**

**classmethod register** (*subcls*)

**values**

**class** `robot.parsing.model.statements.Return` (*tokens, errors=()*)

Bases: `robot.parsing.model.statements.MultiValue`

**type** = `'RETURN'`

**classmethod from\_params** (*args, indent=' ', separator=' ', eol='\n'*)

Create statement from passed parameters.

Required and optional arguments should match class properties. Values are used to create matching tokens.

There is one notable difference for *Documentation* statement where `settings_header` flag is used to determine if statement belongs to settings header or test/keyword.

Most implementations support following general properties: - *separator* whitespace inserted between each token. Default is four spaces. - *indent* whitespace inserted before first token. Default is four spaces. - *eol* end of line sign. Default is `'\n'`.

**col\_offset**

**data\_tokens**

**end\_col\_offset**

```

end_lineno

classmethod from_tokens (tokens)

get_token (*types)
    Return a token with the given type.

    If there are no matches, return None. If there are multiple matches, return the first match.

get_tokens (*types)
    Return tokens having any of the given types.

get_value (type, default=None)
    Return value of a token with the given type.

    If there are no matches, return default. If there are multiple matches, return the value of the first match.

get_values (*types)
    Return values of tokens having any of the given types.

handles_types = ()

lineno

lines

classmethod register (subcls)

validate (context)

values

class robot.parsing.model.statements.KeywordCall (tokens, errors=())
    Bases: robot.parsing.model.statements.Statement

    type = 'KEYWORD'

    classmethod from_params (name, assign=(), args=(), indent=' ', separator=' ', eol='\n')
        Create statement from passed parameters.

        Required and optional arguments should match class properties. Values are used to create matching tokens.

        There is one notable difference for Documentation statement where settings_header flag is used to
        determine if statement belongs to settings header or test/keyword.

        Most implementations support following general properties: - separator whitespace inserted between each
        token. Default is four spaces. - indent whitespace inserted before first token. Default is four spaces. -
        eol end of line sign. Default is '\n'.

    keyword

    args

    assign

    col_offset

    data_tokens

    end_col_offset

    end_lineno

    classmethod from_tokens (tokens)

    get_token (*types)
        Return a token with the given type.

```

If there are no matches, return `None`. If there are multiple matches, return the first match.

**get\_tokens** (\*types)

Return tokens having any of the given types.

**get\_value** (type, default=None)

Return value of a token with the given type.

If there are no matches, return `default`. If there are multiple matches, return the value of the first match.

**get\_values** (\*types)

Return values of tokens having any of the given types.

**handles\_types** = ()

**lineno**

**lines**

**classmethod register** (subcls)

**validate** (context)

**class** robot.parsing.model.statements.**TemplateArguments** (tokens, errors=())

Bases: `robot.parsing.model.statements.Statement`

**type** = 'ARGUMENT'

**classmethod from\_params** (args, indent=' ', separator=' ', eol='\n')

Create statement from passed parameters.

Required and optional arguments should match class properties. Values are used to create matching tokens.

There is one notable difference for *Documentation* statement where `settings_header` flag is used to determine if statement belongs to settings header or test/keyword.

Most implementations support following general properties: - *separator* whitespace inserted between each token. Default is four spaces. - *indent* whitespace inserted before first token. Default is four spaces. - *eol* end of line sign. Default is '`\n`'.

**args**

**col\_offset**

**data\_tokens**

**end\_col\_offset**

**end\_lineno**

**classmethod from\_tokens** (tokens)

**get\_token** (\*types)

Return a token with the given type.

If there are no matches, return `None`. If there are multiple matches, return the first match.

**get\_tokens** (\*types)

Return tokens having any of the given types.

**get\_value** (type, default=None)

Return value of a token with the given type.

If there are no matches, return `default`. If there are multiple matches, return the value of the first match.

**get\_values** (\*types)

Return values of tokens having any of the given types.

```

handles_types = ()
lineno
lines
classmethod register (subcls)
validate (context)

```

**class** robot.parsing.model.statements.**ForHeader** (*tokens, errors=()*)  
 Bases: *robot.parsing.model.statements.Statement*

**type** = 'FOR'

**classmethod from\_params** (*variables, values, flavor='IN', indent=' ', separator=' ', eol='\n'*)  
 Create statement from passed parameters.

Required and optional arguments should match class properties. Values are used to create matching tokens.

There is one notable difference for *Documentation* statement where `settings_header` flag is used to determine if statement belongs to settings header or test/keyword.

Most implementations support following general properties: - *separator* whitespace inserted between each token. Default is four spaces. - *indent* whitespace inserted before first token. Default is four spaces. - *eol* end of line sign. Default is '\n'.

**variables**  
**values**  
**flavor**  
**validate** (*context*)  
**col\_offset**  
**data\_tokens**  
**end\_col\_offset**  
**end\_lineno**  
**classmethod from\_tokens** (*tokens*)  
**get\_token** (*\*types*)  
 Return a token with the given type.  
 If there are no matches, return None. If there are multiple matches, return the first match.  
**get\_tokens** (*\*types*)  
 Return tokens having any of the given types.  
**get\_value** (*type, default=None*)  
 Return value of a token with the given type.  
 If there are no matches, return default. If there are multiple matches, return the value of the first match.  
**get\_values** (*\*types*)  
 Return values of tokens having any of the given types.

```

handles_types = ()
lineno
lines
classmethod register (subcls)

```

```
class robot.parsing.model.statements.IfElseHeader (tokens, errors=())
    Bases: robot.parsing.model.statements.Statement

    condition
    assign
    col_offset
    data_tokens
    end_col_offset
    end_lineno

    classmethod from_params (*args, **kwargs)
        Create statement from passed parameters.

        Required and optional arguments should match class properties. Values are used to create matching tokens.

        There is one notable difference for Documentation statement where settings_header flag is used to
        determine if statement belongs to settings header or test/keyword.

        Most implementations support following general properties: - separator whitespace inserted between each
        token. Default is four spaces. - indent whitespace inserted before first token. Default is four spaces. -
        eol end of line sign. Default is '\n'.

    classmethod from_tokens (tokens)

    get_token (*types)
        Return a token with the given type.

        If there are no matches, return None. If there are multiple matches, return the first match.

    get_tokens (*types)
        Return tokens having any of the given types.

    get_value (type, default=None)
        Return value of a token with the given type.

        If there are no matches, return default. If there are multiple matches, return the value of the first match.

    get_values (*types)
        Return values of tokens having any of the given types.

    handles_types = ()

    lineno
    lines

    classmethod register (subcls)

    type = None

    validate (context)
```

```
class robot.parsing.model.statements.IfHeader (tokens, errors=())
    Bases: robot.parsing.model.statements.IfElseHeader

    type = 'IF'

    classmethod from_params (condition, indent=' ', separator=' ', eol='\n')
        Create statement from passed parameters.

        Required and optional arguments should match class properties. Values are used to create matching tokens.
```

There is one notable difference for *Documentation* statement where `settings_header` flag is used to determine if statement belongs to settings header or test/keyword.

Most implementations support following general properties: - *separator* whitespace inserted between each token. Default is four spaces. - *indent* whitespace inserted before first token. Default is four spaces. - *eol* end of line sign. Default is `'\n'`.

**condition**

**validate** (*context*)

**assign**

**col\_offset**

**data\_tokens**

**end\_col\_offset**

**end\_lineno**

**classmethod from\_tokens** (*tokens*)

**get\_token** (*\*types*)

Return a token with the given type.

If there are no matches, return None. If there are multiple matches, return the first match.

**get\_tokens** (*\*types*)

Return tokens having any of the given types.

**get\_value** (*type, default=None*)

Return value of a token with the given type.

If there are no matches, return default. If there are multiple matches, return the value of the first match.

**get\_values** (*\*types*)

Return values of tokens having any of the given types.

**handles\_types** = ()

**lineno**

**lines**

**classmethod register** (*subcls*)

**class** robot.parsing.model.statements.**InlineIfHeader** (*tokens, errors=()*)

Bases: `robot.parsing.model.statements.IfHeader`

**type** = 'INLINE IF'

**assign**

**col\_offset**

**condition**

**data\_tokens**

**end\_col\_offset**

**end\_lineno**

**classmethod from\_params** (*condition, indent=' ', separator=' ', eol='\n'*)

Create statement from passed parameters.

Required and optional arguments should match class properties. Values are used to create matching tokens.

There is one notable difference for *Documentation* statement where `settings_header` flag is used to determine if statement belongs to settings header or test/keyword.

Most implementations support following general properties: - *separator* whitespace inserted between each token. Default is four spaces. - *indent* whitespace inserted before first token. Default is four spaces. - *eol* end of line sign. Default is `'\n'`.

**classmethod** `from_tokens` (*tokens*)

**get\_token** (*\*types*)

Return a token with the given type.

If there are no matches, return `None`. If there are multiple matches, return the first match.

**get\_tokens** (*\*types*)

Return tokens having any of the given types.

**get\_value** (*type, default=None*)

Return value of a token with the given type.

If there are no matches, return `default`. If there are multiple matches, return the value of the first match.

**get\_values** (*\*types*)

Return values of tokens having any of the given types.

**handles\_types** = ()

**lineno**

**lines**

**classmethod** `register` (*subcls*)

**validate** (*context*)

**class** `robot.parsing.model.statements.ElseIfHeader` (*tokens, errors=()*)

Bases: `robot.parsing.model.statements.IfHeader`

**type** = 'ELSE IF'

**assign**

**col\_offset**

**condition**

**data\_tokens**

**end\_col\_offset**

**end\_lineno**

**classmethod** `from_params` (*condition, indent=' ', separator=' ', eol='\n'*)

Create statement from passed parameters.

Required and optional arguments should match class properties. Values are used to create matching tokens.

There is one notable difference for *Documentation* statement where `settings_header` flag is used to determine if statement belongs to settings header or test/keyword.

Most implementations support following general properties: - *separator* whitespace inserted between each token. Default is four spaces. - *indent* whitespace inserted before first token. Default is four spaces. - *eol* end of line sign. Default is `'\n'`.

**classmethod** `from_tokens` (*tokens*)



```

get_token (*types)
    Return a token with the given type.

    If there are no matches, return None. If there are multiple matches, return the first match.

get_tokens (*types)
    Return tokens having any of the given types.

get_value (type, default=None)
    Return value of a token with the given type.

    If there are no matches, return default. If there are multiple matches, return the value of the first match.

get_values (*types)
    Return values of tokens having any of the given types.

handles_types = ()

lineno

lines

classmethod register (subcls)

validate (context)

class robot.parsing.model.statements.ElseHeader (tokens, errors=())
    Bases: robot.parsing.model.statements.IfElseHeader

    type = 'ELSE'

    classmethod from_params (indent=' ', eol='\n')
        Create statement from passed parameters.

        Required and optional arguments should match class properties. Values are used to create matching tokens.

        There is one notable difference for Documentation statement where settings_header flag is used to
        determine if statement belongs to settings header or test/keyword.

        Most implementations support following general properties: - separator whitespace inserted between each
        token. Default is four spaces. - indent whitespace inserted before first token. Default is four spaces. -
        eol end of line sign. Default is '\n'.

    validate (context)

    assign

    col_offset

    condition

    data_tokens

    end_col_offset

    end_lineno

    classmethod from_tokens (tokens)

    get_token (*types)
        Return a token with the given type.

        If there are no matches, return None. If there are multiple matches, return the first match.

    get_tokens (*types)
        Return tokens having any of the given types.

```

**get\_value** (*type*, *default=None*)  
Return value of a token with the given *type*.  
If there are no matches, return *default*. If there are multiple matches, return the value of the first match.

**get\_values** (*\*types*)  
Return values of tokens having any of the given *types*.

**handles\_types** = ()

**lineno**

**lines**

**classmethod register** (*subcls*)

**class** robot.parsing.model.statements.NoArgumentHeader (*tokens*, *errors=()*)  
Bases: *robot.parsing.model.statements.Statement*

**classmethod from\_params** (*indent=' ', eol='\n'*)  
Create statement from passed parameters.  
Required and optional arguments should match class properties. Values are used to create matching tokens.  
There is one notable difference for *Documentation* statement where *settings\_header* flag is used to determine if statement belongs to settings header or test/keyword.  
Most implementations support following general properties: - *separator* whitespace inserted between each token. Default is four spaces. - *indent* whitespace inserted before first token. Default is four spaces. - *eol* end of line sign. Default is '*\n*'.

**validate** (*context*)

**values**

**col\_offset**

**data\_tokens**

**end\_col\_offset**

**end\_lineno**

**classmethod from\_tokens** (*tokens*)

**get\_token** (*\*types*)  
Return a token with the given *type*.  
If there are no matches, return *None*. If there are multiple matches, return the first match.

**get\_tokens** (*\*types*)  
Return tokens having any of the given *types*.

**get\_value** (*type*, *default=None*)  
Return value of a token with the given *type*.  
If there are no matches, return *default*. If there are multiple matches, return the value of the first match.

**get\_values** (*\*types*)  
Return values of tokens having any of the given *types*.

**handles\_types** = ()

**lineno**

**lines**

**classmethod register** (*subcls*)

```

    type = None
class robot.parsing.model.statements.TryHeader(tokens, errors=())
    Bases: robot.parsing.model.statements.NoArgumentHeader
    type = 'TRY'
    col_offset
    data_tokens
    end_col_offset
    end_lineno
    classmethod from_params(indent=' ', eol='\n')
        Create statement from passed parameters.

        Required and optional arguments should match class properties. Values are used to create matching tokens.

        There is one notable difference for Documentation statement where settings_header flag is used to
        determine if statement belongs to settings header or test/keyword.

        Most implementations support following general properties: - separator whitespace inserted between each
        token. Default is four spaces. - indent whitespace inserted before first token. Default is four spaces. -
        eol end of line sign. Default is '\n'.

    classmethod from_tokens(tokens)
    get_token(*types)
        Return a token with the given type.

        If there are no matches, return None. If there are multiple matches, return the first match.
    get_tokens(*types)
        Return tokens having any of the given types.
    get_value(type, default=None)
        Return value of a token with the given type.

        If there are no matches, return default. If there are multiple matches, return the value of the first match.
    get_values(*types)
        Return values of tokens having any of the given types.
    handles_types = ()
    lineno
    lines
    classmethod register(subcls)
    validate(context)
    values

class robot.parsing.model.statements.ExceptHeader(tokens, errors=())
    Bases: robot.parsing.model.statements.Statement
    type = 'EXCEPT'
    classmethod from_params(patterns=None, variable=None, indent=' ', separator=' ', eol='\n')
        Create statement from passed parameters.

        Required and optional arguments should match class properties. Values are used to create matching tokens.

```

There is one notable difference for *Documentation* statement where `settings_header` flag is used to determine if statement belongs to settings header or test/keyword.

Most implementations support following general properties: - *separator* whitespace inserted between each token. Default is four spaces. - *indent* whitespace inserted before first token. Default is four spaces. - *eol* end of line sign. Default is `'\n'`.

**patterns**

**pattern\_type**

**variable**

**validate** (*context*)

**col\_offset**

**data\_tokens**

**end\_col\_offset**

**end\_lineno**

**classmethod from\_tokens** (*tokens*)

**get\_token** (*\*types*)

Return a token with the given type.

If there are no matches, return `None`. If there are multiple matches, return the first match.

**get\_tokens** (*\*types*)

Return tokens having any of the given types.

**get\_value** (*type, default=None*)

Return value of a token with the given type.

If there are no matches, return `default`. If there are multiple matches, return the value of the first match.

**get\_values** (*\*types*)

Return values of tokens having any of the given types.

**handles\_types** = ()

**lineno**

**lines**

**classmethod register** (*subcls*)

**class** robot.parsing.model.statements.FinallyHeader (*tokens, errors=()*)

Bases: `robot.parsing.model.statements.NoArgumentHeader`

**type** = 'FINALLY'

**col\_offset**

**data\_tokens**

**end\_col\_offset**

**end\_lineno**

**classmethod from\_params** (*indent=' ', eol='\n'*)

Create statement from passed parameters.

Required and optional arguments should match class properties. Values are used to create matching tokens.

There is one notable difference for *Documentation* statement where `settings_header` flag is used to determine if statement belongs to settings header or test/keyword.

Most implementations support following general properties: - *separator* whitespace inserted between each token. Default is four spaces. - *indent* whitespace inserted before first token. Default is four spaces. - *eol* end of line sign. Default is `'\n'`.

**classmethod** `from_tokens` (*tokens*)

**get\_token** (*\*types*)

Return a token with the given `type`.

If there are no matches, return `None`. If there are multiple matches, return the first match.

**get\_tokens** (*\*types*)

Return tokens having any of the given `types`.

**get\_value** (*type, default=None*)

Return value of a token with the given `type`.

If there are no matches, return `default`. If there are multiple matches, return the value of the first match.

**get\_values** (*\*types*)

Return values of tokens having any of the given `types`.

**handles\_types** = ()

**lineno**

**lines**

**classmethod** `register` (*subcls*)

**validate** (*context*)

**values**

**class** `robot.parsing.model.statements.End` (*tokens, errors=()*)

Bases: `robot.parsing.model.statements.NoArgumentHeader`

**type** = 'END'

**col\_offset**

**data\_tokens**

**end\_col\_offset**

**end\_lineno**

**classmethod** `from_params` (*indent=' ', eol='\n'*)

Create statement from passed parameters.

Required and optional arguments should match class properties. Values are used to create matching tokens.

There is one notable difference for *Documentation* statement where `settings_header` flag is used to determine if statement belongs to settings header or test/keyword.

Most implementations support following general properties: - *separator* whitespace inserted between each token. Default is four spaces. - *indent* whitespace inserted before first token. Default is four spaces. - *eol* end of line sign. Default is `'\n'`.

**classmethod** `from_tokens` (*tokens*)

**get\_token** (*\*types*)

Return a token with the given `type`.

If there are no matches, return `None`. If there are multiple matches, return the first match.

**get\_tokens** (\*types)

Return tokens having any of the given types.

**get\_value** (type, default=None)

Return value of a token with the given type.

If there are no matches, return `default`. If there are multiple matches, return the value of the first match.

**get\_values** (\*types)

Return values of tokens having any of the given types.

**handles\_types** = ()

**lineno**

**lines**

**classmethod register** (subcls)

**validate** (context)

**values**

**class** robot.parsing.model.statements.**WhileHeader** (tokens, errors=())

Bases: [robot.parsing.model.statements.Statement](#)

**type** = 'WHILE'

**classmethod from\_params** (condition, limit=None, indent=' ', separator=' ', eol='\n')

Create statement from passed parameters.

Required and optional arguments should match class properties. Values are used to create matching tokens.

There is one notable difference for *Documentation* statement where `settings_header` flag is used to determine if statement belongs to settings header or test/keyword.

Most implementations support following general properties: - *separator* whitespace inserted between each token. Default is four spaces. - *indent* whitespace inserted before first token. Default is four spaces. - *eol* end of line sign. Default is `'\n'`.

**condition**

**limit**

**validate** (context)

**col\_offset**

**data\_tokens**

**end\_col\_offset**

**end\_lineno**

**classmethod from\_tokens** (tokens)

**get\_token** (\*types)

Return a token with the given type.

If there are no matches, return `None`. If there are multiple matches, return the first match.

**get\_tokens** (\*types)

Return tokens having any of the given types.

```

get_value (type, default=None)
    Return value of a token with the given type.

    If there are no matches, return default. If there are multiple matches, return the value of the first match.

get_values (*types)
    Return values of tokens having any of the given types.

handles_types = ()

lineno

lines

classmethod register (subcls)

class robot.parsing.model.statements.ReturnStatement (tokens, errors=())
    Bases: robot.parsing.model.statements.Statement

    type = 'RETURN STATEMENT'

    values

    classmethod from_params (values=(), indent=' ', separator=' ', eol='\n')
        Create statement from passed parameters.

        Required and optional arguments should match class properties. Values are used to create matching tokens.

        There is one notable difference for Documentation statement where settings_header flag is used to
        determine if statement belongs to settings header or test/keyword.

        Most implementations support following general properties: - separator whitespace inserted between each
        token. Default is four spaces. - indent whitespace inserted before first token. Default is four spaces. -
        eol end of line sign. Default is '\n'.

    validate (context)

    col_offset

    data_tokens

    end_col_offset

    end_lineno

    classmethod from_tokens (tokens)

    get_token (*types)
        Return a token with the given type.

        If there are no matches, return None. If there are multiple matches, return the first match.

    get_tokens (*types)
        Return tokens having any of the given types.

    get_value (type, default=None)
        Return value of a token with the given type.

        If there are no matches, return default. If there are multiple matches, return the value of the first match.

    get_values (*types)
        Return values of tokens having any of the given types.

    handles_types = ()

    lineno

    lines

```

```
classmethod register(subcls)

class robot.parsing.model.statements.LoopControl(tokens, errors=())
    Bases: robot.parsing.model.statements.NoArgumentHeader
    validate(context)
    col_offset
    data_tokens
    end_col_offset
    end_lineno

classmethod from_params(indent=' ', eol='\n')
    Create statement from passed parameters.

    Required and optional arguments should match class properties. Values are used to create matching tokens.

    There is one notable difference for Documentation statement where settings_header flag is used to
    determine if statement belongs to settings header or test/keyword.

    Most implementations support following general properties: - separator whitespace inserted between each
    token. Default is four spaces. - indent whitespace inserted before first token. Default is four spaces. -
    eol end of line sign. Default is '\n'.

classmethod from_tokens(tokens)

get_token(*types)
    Return a token with the given type.

    If there are no matches, return None. If there are multiple matches, return the first match.

get_tokens(*types)
    Return tokens having any of the given types.

get_value(type, default=None)
    Return value of a token with the given type.

    If there are no matches, return default. If there are multiple matches, return the value of the first match.

get_values(*types)
    Return values of tokens having any of the given types.

handles_types = ()

lineno

lines

classmethod register(subcls)

type = None

values

class robot.parsing.model.statements.Continue(tokens, errors=())
    Bases: robot.parsing.model.statements.LoopControl
    type = 'CONTINUE'
    col_offset
    data_tokens
    end_col_offset
```



**end\_lineno**

**classmethod from\_params** (*indent=' ', eol='\n'*)

Create statement from passed parameters.

Required and optional arguments should match class properties. Values are used to create matching tokens.

There is one notable difference for *Documentation* statement where `settings_header` flag is used to determine if statement belongs to settings header or test/keyword.

Most implementations support following general properties: - *separator* whitespace inserted between each token. Default is four spaces. - *indent* whitespace inserted before first token. Default is four spaces. - *eol* end of line sign. Default is `'\n'`.

**classmethod from\_tokens** (*tokens*)

**get\_token** (*\*types*)

Return a token with the given type.

If there are no matches, return `None`. If there are multiple matches, return the first match.

**get\_tokens** (*\*types*)

Return tokens having any of the given types.

**get\_value** (*type, default=None*)

Return value of a token with the given type.

If there are no matches, return `default`. If there are multiple matches, return the value of the first match.

**get\_values** (*\*types*)

Return values of tokens having any of the given types.

**handles\_types** = ()

**lineno**

**lines**

**classmethod register** (*subcls*)

**validate** (*context*)

**values**

**class** `robot.parsing.model.statements.Break` (*tokens, errors=()*)

Bases: `robot.parsing.model.statements.LoopControl`

**type** = 'BREAK'

**col\_offset**

**data\_tokens**

**end\_col\_offset**

**end\_lineno**

**classmethod from\_params** (*indent=' ', eol='\n'*)

Create statement from passed parameters.

Required and optional arguments should match class properties. Values are used to create matching tokens.

There is one notable difference for *Documentation* statement where `settings_header` flag is used to determine if statement belongs to settings header or test/keyword.

Most implementations support following general properties: - *separator* whitespace inserted between each token. Default is four spaces. - *indent* whitespace inserted before first token. Default is four spaces. - *eol* end of line sign. Default is '`\n`'.

**classmethod** `from_tokens` (*tokens*)

**get\_token** (*\*types*)

Return a token with the given *type*.

If there are no matches, return `None`. If there are multiple matches, return the first match.

**get\_tokens** (*\*types*)

Return tokens having any of the given *types*.

**get\_value** (*type*, *default=None*)

Return value of a token with the given *type*.

If there are no matches, return *default*. If there are multiple matches, return the value of the first match.

**get\_values** (*\*types*)

Return values of tokens having any of the given *types*.

**handles\_types** = ()

**lineno**

**lines**

**classmethod** `register` (*subcls*)

**validate** (*context*)

**values**

**class** `robot.parsing.model.statements.Comment` (*tokens*, *errors=()*)

Bases: `robot.parsing.model.statements.Statement`

**type** = 'COMMENT'

**classmethod** `from_params` (*comment*, *indent=' '*, *eol='\n'*)

Create statement from passed parameters.

Required and optional arguments should match class properties. Values are used to create matching tokens.

There is one notable difference for *Documentation* statement where `settings_header` flag is used to determine if statement belongs to settings header or test/keyword.

Most implementations support following general properties: - *separator* whitespace inserted between each token. Default is four spaces. - *indent* whitespace inserted before first token. Default is four spaces. - *eol* end of line sign. Default is '`\n`'.

**col\_offset**

**data\_tokens**

**end\_col\_offset**

**end\_lineno**

**classmethod** `from_tokens` (*tokens*)

**get\_token** (*\*types*)

Return a token with the given *type*.

If there are no matches, return `None`. If there are multiple matches, return the first match.

```

get_tokens (*types)
    Return tokens having any of the given types.

get_value (type, default=None)
    Return value of a token with the given type.

    If there are no matches, return default. If there are multiple matches, return the value of the first match.

get_values (*types)
    Return values of tokens having any of the given types.

handles_types = ()

lineno

lines

classmethod register (subcls)

validate (context)

class robot.parsing.model.statements.Error (tokens, errors=())
    Bases: robot.parsing.model.statements.Statement

    type = 'ERROR'

    handles_types = ('ERROR', 'FATAL_ERROR')

    errors
        Errors got from the underlying ERROR and FATAL_ERROR tokens.

        Errors can be set also explicitly. When accessing errors, they are returned along with errors got from
        tokens.

    col_offset

    data_tokens

    end_col_offset

    end_lineno

    classmethod from_params (*args, **kwargs)
        Create statement from passed parameters.

        Required and optional arguments should match class properties. Values are used to create matching tokens.

        There is one notable difference for Documentation statement where settings_header flag is used to
        determine if statement belongs to settings header or test/keyword.

        Most implementations support following general properties: - separator whitespace inserted between each
        token. Default is four spaces. - indent whitespace inserted before first token. Default is four spaces. -
        eol end of line sign. Default is '\n'.

    classmethod from_tokens (tokens)

    get_token (*types)
        Return a token with the given type.

        If there are no matches, return None. If there are multiple matches, return the first match.

    get_tokens (*types)
        Return tokens having any of the given types.

    get_value (type, default=None)
        Return value of a token with the given type.

```

If there are no matches, return `default`. If there are multiple matches, return the value of the first match.

**get\_values** (\*types)

Return values of tokens having any of the given types.

**lineno**

**lines**

**classmethod register** (subcls)

**validate** (context)

**class** robot.parsing.model.statements.**EmptyLine** (tokens, errors=())

Bases: `robot.parsing.model.statements.Statement`

**type** = 'EOL'

**col\_offset**

**data\_tokens**

**end\_col\_offset**

**end\_lineno**

**classmethod from\_params** (eol='\n')

Create statement from passed parameters.

Required and optional arguments should match class properties. Values are used to create matching tokens.

There is one notable difference for *Documentation* statement where `settings_header` flag is used to determine if statement belongs to settings header or test/keyword.

Most implementations support following general properties: - *separator* whitespace inserted between each token. Default is four spaces. - *indent* whitespace inserted before first token. Default is four spaces. - *eol* end of line sign. Default is `'\n'`.

**classmethod from\_tokens** (tokens)

**get\_token** (\*types)

Return a token with the given type.

If there are no matches, return `None`. If there are multiple matches, return the first match.

**get\_tokens** (\*types)

Return tokens having any of the given types.

**get\_value** (type, default=None)

Return value of a token with the given type.

If there are no matches, return `default`. If there are multiple matches, return the value of the first match.

**get\_values** (\*types)

Return values of tokens having any of the given types.

**handles\_types** = ()

**lineno**

**lines**

**classmethod register** (subcls)

**validate** (context)

## robot.parsing.model.visitor module

**class** robot.parsing.model.visitor.**VisitorFinder**

Bases: object

**class** robot.parsing.model.visitor.**ModelVisitor**

Bases: ast.NodeVisitor, [robot.parsing.model.visitor.VisitorFinder](#)

NodeVisitor that supports matching nodes based on their base classes.

Otherwise identical to the standard [ast.NodeVisitor](#), but allows creating `visit_ClassName` methods so that the `ClassName` is one of the base classes of the node. For example, this visitor method matches all statements:

```
def visit_Statement(self, node):
    # ...
```

**visit** (*node*)

Visit a node.

**generic\_visit** (*node*)

Called if no explicit visitor function exists for a node.

**class** robot.parsing.model.visitor.**ModelTransformer**

Bases: ast.NodeTransformer, [robot.parsing.model.visitor.VisitorFinder](#)

NodeTransformer that supports matching nodes based on their base classes.

See [ModelVisitor](#) for explanation how this is different compared to the standard [ast.NodeTransformer](#).

**visit** (*node*)

Visit a node.

**generic\_visit** (*node*)

Called if no explicit visitor function exists for a node.

## robot.parsing.parser package

### Submodules

## robot.parsing.parser.blockparsers module

**class** robot.parsing.parser.blockparsers.**Parser** (*model*)

Bases: object

Base class for parsers.

**handles** (*statement*)

**parse** (*statement*)

**class** robot.parsing.parser.blockparsers.**BlockParser** (*model*)

Bases: [robot.parsing.parser.blockparsers.Parser](#)

**unhandled\_tokens** = frozenset({'COMMENT HEADER', 'TESTCASE NAME', 'KEYWORD HEADER', 'TE

**handles** (*statement*)

**parse** (*statement*)

**class** robot.parsing.parser.blockparsers.**TestCaseParser** (*header*)

Bases: [robot.parsing.parser.blockparsers.BlockParser](#)

```
    handles (statement)
    parse (statement)
    unhandled_tokens = frozenset({'COMMENT HEADER', 'TESTCASE NAME', 'KEYWORD HEADER', 'TE

class robot.parsing.parser.blockparsers.KeywordParser(header)
    Bases: robot.parsing.parser.blockparsers.BlockParser
    handles (statement)
    parse (statement)
    unhandled_tokens = frozenset({'COMMENT HEADER', 'TESTCASE NAME', 'KEYWORD HEADER', 'TE

class robot.parsing.parser.blockparsers.NestedBlockParser(model)
    Bases: robot.parsing.parser.blockparsers.BlockParser
    handles (statement)
    parse (statement)
    unhandled_tokens = frozenset({'COMMENT HEADER', 'TESTCASE NAME', 'KEYWORD HEADER', 'TE

class robot.parsing.parser.blockparsers.ForParser(header)
    Bases: robot.parsing.parser.blockparsers.NestedBlockParser
    handles (statement)
    parse (statement)
    unhandled_tokens = frozenset({'COMMENT HEADER', 'TESTCASE NAME', 'KEYWORD HEADER', 'TE

class robot.parsing.parser.blockparsers.IfParser(header, handle_end=True)
    Bases: robot.parsing.parser.blockparsers.NestedBlockParser
    parse (statement)
    handles (statement)
    unhandled_tokens = frozenset({'COMMENT HEADER', 'TESTCASE NAME', 'KEYWORD HEADER', 'TE

class robot.parsing.parser.blockparsers.TryParser(header, handle_end=True)
    Bases: robot.parsing.parser.blockparsers.NestedBlockParser
    parse (statement)
    handles (statement)
    unhandled_tokens = frozenset({'COMMENT HEADER', 'TESTCASE NAME', 'KEYWORD HEADER', 'TE

class robot.parsing.parser.blockparsers.WhileParser(header)
    Bases: robot.parsing.parser.blockparsers.NestedBlockParser
    handles (statement)
    parse (statement)
    unhandled_tokens = frozenset({'COMMENT HEADER', 'TESTCASE NAME', 'KEYWORD HEADER', 'TE
```

### robot.parsing.parser.fileparser module

```
class robot.parsing.parser.fileparser.FileParser(source=None)
    Bases: robot.parsing.parser.blockparsers.Parser
    handles (statement)
```

```
    parse (statement)

class robot.parsing.parser.fileparser.SectionParser (header)
    Bases: robot.parsing.parser.blockparsers.Parser

    model_class = None

    handles (statement)

    parse (statement)

class robot.parsing.parser.fileparser.SettingSectionParser (header)
    Bases: robot.parsing.parser.fileparser.SectionParser

    model_class
        alias of robot.parsing.model.blocks.SettingSection

    handles (statement)

    parse (statement)

class robot.parsing.parser.fileparser.VariableSectionParser (header)
    Bases: robot.parsing.parser.fileparser.SectionParser

    model_class
        alias of robot.parsing.model.blocks.VariableSection

    handles (statement)

    parse (statement)

class robot.parsing.parser.fileparser.CommentSectionParser (header)
    Bases: robot.parsing.parser.fileparser.SectionParser

    model_class
        alias of robot.parsing.model.blocks.CommentSection

    handles (statement)

    parse (statement)

class robot.parsing.parser.fileparser.ImplicitCommentSectionParser (header)
    Bases: robot.parsing.parser.fileparser.SectionParser

    model_class (statement)

    handles (statement)

    parse (statement)

class robot.parsing.parser.fileparser.TestCaseSectionParser (header)
    Bases: robot.parsing.parser.fileparser.SectionParser

    model_class
        alias of robot.parsing.model.blocks.TestCaseSection

    parse (statement)

    handles (statement)

class robot.parsing.parser.fileparser.KeywordSectionParser (header)
    Bases: robot.parsing.parser.fileparser.SectionParser

    model_class
        alias of robot.parsing.model.blocks.KeywordSection

    parse (statement)
```

**handles** (*statement*)

## robot.parsing.parser.parser module

`robot.parsing.parser.parser.get_model(source, data_only=False, curdir=None)`

Parses the given source to a model represented as an AST.

How to use the model is explained more thoroughly in the general documentation of the `robot.parsing` module.

### Parameters

- **source** – The source where to read the data. Can be a path to a source file as a string or as `pathlib.Path` object, an already opened file object, or Unicode text containing the data directly. Source files must be UTF-8 encoded.
- **data\_only** – When `False` (default), returns all tokens. When set to `True`, omits separators, comments, continuation markers, and other non-data tokens. Model like this cannot be saved back to file system.
- **curdir** – Directory where the source file exists. This path is used to set the value of the built-in `{CURDIR}` variable during parsing. When not given, the variable is left as-is. Should only be given only if the model will be executed afterwards. If the model is saved back to disk, resolving `{CURDIR}` is typically not a good idea.

Use `get_resource_model()` or `get_init_model()` when parsing resource or suite initialization files, respectively.

`robot.parsing.parser.parser.get_resource_model(source, data_only=False, curdir=None)`

Parses the given source to a resource file model.

Otherwise same as `get_model()` but the source is considered to be a resource file. This affects, for example, what settings are valid.

`robot.parsing.parser.parser.get_init_model(source, data_only=False, curdir=None)`

Parses the given source to a init file model.

Otherwise same as `get_model()` but the source is considered to be a suite initialization file. This affects, for example, what settings are valid.

## Submodules

### 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(*items)
```

### robot.reporting.jsexecutionresult module

```
class robot.reporting.jsexecutionresult.JsExecutionResult(suite,          statistics,
                                                         errors,          strings,
                                                         basemillis=None,
                                                         split_results=None,
                                                         min_level=None,    ex-
                                                         pand_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,          ex-
                                                      pand_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(item, split=False)

build_keyword(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 a message starts.

        By default, calls start_body_item() which, by default, does nothing.

        Can return explicit False to stop visiting.
```

**close()**

**end\_result** (*result*)

**end\_body\_item** (*item*)

Called, by default, when keywords, messages or control structures end.

More specific `end_keyword()`, `end_message()`, `:meth:'end_for'`, etc. can be implemented to visit only keywords, messages or specific control structures.

Default implementation does nothing.

**end\_break** (*break\_*)

Called when a BREAK element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_continue** (*continue\_*)

Called when a CONTINUE element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_errors** (*errors=None*)

**end\_for** (*for\_*)

Called when a FOR loop ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_for\_iteration** (*iteration*)

Called when a FOR loop iteration ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_if** (*if\_*)

Called when an IF/ELSE structure ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_if\_branch** (*branch*)

Called when an IF/ELSE branch ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_keyword** (*kw*)

Called when a keyword ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_message** (*msg*)

Called when a message ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_return** (*return\_*)

Called when a RETURN element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_stat** (*stat*)

**end\_statistics** (*stats*)

**end\_suite** (*suite*)

Called when a suite ends. Default implementation does nothing.

**end\_suite\_statistics** (*tag\_stats*)

**end\_tag\_statistics** (*tag\_stats*)

**end\_test** (*test*)

Called when a test ends. Default implementation does nothing.

**end\_total\_statistics** (*total\_stats*)

**end\_try** (*root*)

Called when a TRY/EXCEPT structure ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_try\_branch** (*branch*)

Called when TRY, EXCEPT, ELSE and FINALLY branches end.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_while** (*while\_*)

Called when a WHILE loop ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_while\_iteration** (*iteration*)

Called when a WHILE loop iteration ends.

By default, calls `end_body_item()` which, by default, does nothing.

**log\_message** (*msg*)

**message** (*msg*)

**set\_log\_level** (*level*)

**start\_body\_item** (*item*)

Called, by default, when keywords, messages or control structures start.

More specific `start_keyword()`, `start_message()`, `:meth:'start_for`, etc. can be implemented to visit only keywords, messages or specific control structures.

Can return explicit `False` to stop visiting. Default implementation does nothing.

**start\_break** (*break\_*)

Called when a BREAK element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_continue** (*continue\_*)

Called when a CONTINUE element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_errors** (*errors=None*)

**start\_for** (*for\_*)

Called when a FOR loop starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_for\_iteration** (*iteration*)

Called when a FOR loop iteration starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_if** (*if\_*)

Called when an IF/ELSE structure starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_if\_branch** (*branch*)

Called when an IF/ELSE branch starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_keyword** (*kw*)

Called when a keyword starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_result** (*result*)

**start\_return** (*return\_*)

Called when a RETURN element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_stat** (*stat*)

**start\_statistics** (*stats*)

**start\_suite** (*suite*)

Called when a 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 a test starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

**start\_total\_statistics** (*total\_stats*)

**start\_try** (*root*)

Called when a TRY/EXCEPT structure starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_try\_branch** (*branch*)

Called when TRY, EXCEPT, ELSE or FINALLY branches start.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_while** (*while\_*)

Called when a WHILE loop starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_while\_iteration** (*iteration*)

Called when a WHILE loop iteration starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**visit\_break** (*break\_*)

Visits BREAK elements.

**visit\_continue** (*continue\_*)

Visits CONTINUE elements.

**visit\_errors** (*errors*)**visit\_for** (*for\_*)

Implements traversing through FOR loops.

Can be overridden to allow modifying the passed in `for_` without calling `start_for()` or `end_for()` nor visiting body.

**visit\_for\_iteration** (*iteration*)

Implements traversing through single FOR loop iteration.

This is only used with the result side model because on the running side there are no iterations.

Can be overridden to allow modifying the passed in `iteration` without calling `start_for_iteration()` or `end_for_iteration()` nor visiting body.

**visit\_if** (*if\_*)

Implements traversing through IF/ELSE structures.

Notice that `if_` does not have any data directly. Actual IF/ELSE branches are in its body and visited using `visit_if_branch()`.

Can be overridden to allow modifying the passed in `if_` without calling `start_if()` or `end_if()` nor visiting branches.

**visit\_if\_branch** (*branch*)

Implements traversing through single IF/ELSE branch.

Can be overridden to allow modifying the passed in `branch` without calling `start_if_branch()` or `end_if_branch()` nor visiting body.

**visit\_keyword** (*kw*)

Implements traversing through keywords.

Can be overridden to allow modifying the passed in `kw` without calling `start_keyword()` or `end_keyword()` nor visiting the body of the keyword

**visit\_message** (*msg*)

Implements visiting messages.

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

**visit\_result** (*result*)

**visit\_return** (*return\_*)

Visits a RETURN elements.

**visit\_stat** (*stat*)

**visit\_statistics** (*stats*)

**visit\_suite** (*suite*)

Implements traversing through suites.

Can be overridden to allow modifying the passed in *suite* without calling *start\_suite()* or *end\_suite()* nor visiting child suites, tests or setup and teardown at all.

**visit\_suite\_statistics** (*stats*)

**visit\_tag\_statistics** (*stats*)

**visit\_test** (*test*)

Implements traversing through tests.

Can be overridden to allow modifying the passed in *test* without calling *start\_test()* or *end\_test()* nor visiting the body of the test.

**visit\_total\_statistics** (*stats*)

**visit\_try** (*try\_*)

Implements traversing through TRY/EXCEPT structures.

This method is used with the TRY/EXCEPT root element. Actual TRY, EXCEPT, ELSE and FINALLY branches are visited separately using *visit\_try\_branch()*.

**visit\_try\_branch** (*branch*)

Visits individual TRY, EXCEPT, ELSE and FINALLY branches.

**visit\_while** (*while\_*)

Implements traversing through WHILE loops.

Can be overridden to allow modifying the passed in *while\_* without calling *start\_while()* or *end\_while()* nor visiting body.

**visit\_while\_iteration** (*iteration*)

Implements traversing through single WHILE loop iteration.

This is only used with the result side model because on the running side there are no iterations.

Can be overridden to allow modifying the passed in *iteration* without calling *start\_while\_iteration()* or *end\_while\_iteration()* nor visiting body.

## robot.reporting.resultwriter module

**class** robot.reporting.resultwriter.**ResultWriter** (\**sources*)

Bases: object

A class to create log, report, output XML and xUnit files.

**Parameters** **sources** – Either one *Result* object, or one or more paths to existing output XML files.

By default writes *report.html* and *log.html*, but no output XML or xUnit files. Custom file names can be given and results disabled or enabled using settings or options passed to the *write\_results()* method. The latter is typically more convenient:



```
writer = ResultWriter(result)
writer.write_results(report='custom.html', log=None, xunit='xunit.xml')
```

**write\_results** (*settings=None, \*\*options*)

Writes results based on the given settings or options.

**Parameters**

- **settings** – *RebotSettings* object to configure result writing.
- **options** – Used to construct new *RebotSettings* object if settings are not given.

**class** robot.reporting.resultwriter.**Results** (*settings, \*sources*)

Bases: object

**result**

**js\_result**

### robot.reporting.stringcache module

**class** robot.reporting.stringcache.**StringIndex**

Bases: int

**bit\_length** ()

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

**from\_bytes** ()

Return the integer represented by the given array of bytes.

**bytes** Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

**byteorder** The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use 'sys.byteorder' as the byte order value.

**signed** Indicates whether two's complement is used to represent the integer.

**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

**to\_bytes()**

Return an array of bytes representing an integer.

**length** Length of bytes object to use. An `OverflowError` is raised if the integer is not representable with the given number of bytes.

**byteorder** The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use 'sys.byteorder' as the byte order value.

**signed** Determines whether two's complement is used to represent the integer. If signed is False and a negative integer is given, an `OverflowError` is raised.

**class** `robot.reporting.stringcache.StringCache`

Bases: `object`

**add** (*text*, *html=False*)

**dump** ()

### `robot.reporting.xunitwriter` module

**class** `robot.reporting.xunitwriter.XUnitWriter` (*execution\_result*)

Bases: `object`

**write** (*output*)

**class** `robot.reporting.xunitwriter.XUnitFileWriter` (*xml\_writer*)

Bases: `robot.result.visitor.ResultVisitor`

Provides an xUnit-compatible result file.

Attempts to adhere to the de facto schema guessed by Peter Reilly, see: <http://marc.info/?l=ant-dev&m=123551933508682>

**start\_suite** (*suite*)

Called when a suite starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

**end\_suite** (*suite*)

Called when a suite ends. Default implementation does nothing.

**visit\_test** (*test*)

Implements traversing through tests.

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

**visit\_keyword** (*kw*)

Implements traversing through keywords.

Can be overridden to allow modifying the passed in `kw` without calling `start_keyword()` or `end_keyword()` nor visiting the body of the keyword

**visit\_statistics** (*stats*)

**visit\_errors** (*errors*)

**end\_result** (*result*)

**end\_body\_item** (*item*)

Called, by default, when keywords, messages or control structures end.

More specific `end_keyword()`, `end_message()`, `:meth:'end_for'`, etc. can be implemented to visit only keywords, messages or specific control structures.

Default implementation does nothing.

**end\_break** (*break\_*)

Called when a BREAK element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_continue** (*continue\_*)

Called when a CONTINUE element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_errors** (*errors*)**end\_for** (*for\_*)

Called when a FOR loop ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_for\_iteration** (*iteration*)

Called when a FOR loop iteration ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_if** (*if\_*)

Called when an IF/ELSE structure ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_if\_branch** (*branch*)

Called when an IF/ELSE branch ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_keyword** (*keyword*)

Called when a keyword ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_message** (*msg*)

Called when a message ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_return** (*return\_*)

Called when a RETURN element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_stat** (*stat*)**end\_statistics** (*stats*)**end\_suite\_statistics** (*suite\_stats*)**end\_tag\_statistics** (*stats*)**end\_test** (*test*)

Called when a test ends. Default implementation does nothing.

**end\_total\_statistics** (*stats*)

**end\_try** (*try\_*)

Called when a TRY/EXCEPT structure ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_try\_branch** (*branch*)

Called when TRY, EXCEPT, ELSE and FINALLY branches end.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_while** (*while\_*)

Called when a WHILE loop ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_while\_iteration** (*iteration*)

Called when a WHILE loop iteration ends.

By default, calls `end_body_item()` which, by default, does nothing.

**start\_body\_item** (*item*)

Called, by default, when keywords, messages or control structures start.

More specific `start_keyword()`, `start_message()`, `:meth:'start_for`, etc. can be implemented to visit only keywords, messages or specific control structures.

Can return explicit `False` to stop visiting. Default implementation does nothing.

**start\_break** (*break\_*)

Called when a BREAK element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_continue** (*continue\_*)

Called when a CONTINUE element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_errors** (*errors*)**start\_for** (*for\_*)

Called when a FOR loop starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_for\_iteration** (*iteration*)

Called when a FOR loop iteration starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_if** (*if\_*)

Called when an IF/ELSE structure starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_if\_branch** (*branch*)

Called when an IF/ELSE branch starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_keyword** (*keyword*)

Called when a keyword starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_message** (*msg*)

Called when a message starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_result** (*result*)**start\_return** (*return\_*)

Called when a RETURN element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_stat** (*stat*)**start\_statistics** (*stats*)**start\_suite\_statistics** (*stats*)**start\_tag\_statistics** (*stats*)**start\_test** (*test*)

Called when a test starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

**start\_total\_statistics** (*stats*)**start\_try** (*try\_*)

Called when a TRY/EXCEPT structure starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_try\_branch** (*branch*)

Called when TRY, EXCEPT, ELSE or FINALLY branches start.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_while** (*while\_*)

Called when a WHILE loop starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_while\_iteration** (*iteration*)

Called when a WHILE loop iteration starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**visit\_break** (*break\_*)

Visits BREAK elements.

**visit\_continue** (*continue\_*)

Visits CONTINUE elements.

**visit\_for** (*for\_*)

Implements traversing through FOR loops.

Can be overridden to allow modifying the passed in `for_` without calling `start_for()` or `end_for()` nor visiting body.

**visit\_for\_iteration** (*iteration*)

Implements traversing through single FOR loop iteration.

This is only used with the result side model because on the running side there are no iterations.

Can be overridden to allow modifying the passed in `iteration` without calling `start_for_iteration()` or `end_for_iteration()` nor visiting body.

**visit\_if** (*if\_*)

Implements traversing through IF/ELSE structures.

Notice that `if_` does not have any data directly. Actual IF/ELSE branches are in its body and visited using `visit_if_branch()`.

Can be overridden to allow modifying the passed in `if_` without calling `start_if()` or `end_if()` nor visiting branches.

**visit\_if\_branch** (*branch*)

Implements traversing through single IF/ELSE branch.

Can be overridden to allow modifying the passed in `branch` without calling `start_if_branch()` or `end_if_branch()` nor visiting body.

**visit\_message** (*msg*)

Implements visiting messages.

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

**visit\_result** (*result*)**visit\_return** (*return\_*)

Visits a RETURN elements.

**visit\_stat** (*stat*)**visit\_suite** (*suite*)

Implements traversing through suites.

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

**visit\_suite\_statistics** (*stats*)**visit\_tag\_statistics** (*stats*)**visit\_total\_statistics** (*stats*)

**visit\_try** (*try\_*)

Implements traversing through TRY/EXCEPT structures.

This method is used with the TRY/EXCEPT root element. Actual TRY, EXCEPT, ELSE and FINALLY branches are visited separately using `visit_try_branch()`.

**visit\_try\_branch** (*branch*)

Visits individual TRY, EXCEPT, ELSE and FINALLY branches.

**visit\_while** (*while\_*)

Implements traversing through WHILE loops.

Can be overridden to allow modifying the passed in `while_` without calling `start_while()` or `end_while()` nor visiting body.

**visit\_while\_iteration** (*iteration*)

Implements traversing through single WHILE loop iteration.

This is only used with the result side model because on the running side there are no iterations.

Can be overridden to allow modifying the passed in `iteration` without calling `start_while_iteration()` or `end_while_iteration()` nor visiting body.

**robot.result package**

Implements parsing execution results from XML output files.

The main public API of this package consists of the `ExecutionResult()` factory method, that returns `Result` objects, and of the `ResultVisitor` abstract class, that eases further processing the results.

The model objects in the `model` module can also be considered to be part of the public API, because they can be found inside the `Result` object. They can also be inspected and modified as part of the normal test execution by `pre-Rebot` modifiers and listeners.

It is highly recommended to import the public entry-points via the `robot.api` package like in the example below. In those rare cases where the aforementioned model objects are needed directly, they can be imported from this package.

This package is considered stable.

**Example**

```
#!/usr/bin/env python

"""Usage: check_test_times.py seconds inpath [outpath]

Reads test execution result from an output XML file and checks that no test
took longer than given amount of seconds to execute.

Optional `outpath` specifies where to write processed results. If not given,
results are written over the original file.
"""

import sys
from robot.api import ExecutionResult, ResultVisitor

class ExecutionTimeChecker(ResultVisitor):
```

(continues on next page)

(continued from previous page)

```
def __init__(self, max_seconds):
    self.max_milliseconds = max_seconds * 1000

def visit_test(self, test):
    if test.status == 'PASS' and test.elapsedtime > self.max_milliseconds:
        test.status = 'FAIL'
        test.message = 'Test execution took too long.'

def check_tests(seconds, inpath, outpath=None):
    result = ExecutionResult(inpath)
    result.visit(ExecutionTimeChecker(float(seconds)))
    result.save(outpath)

if __name__ == '__main__':
    try:
        check_tests(*sys.argv[1:])
    except TypeError:
        print(__doc__)
```

## Submodules

### robot.result.configurer module

```
class robot.result.configurer.SuiteConfigurer(remove_keywords=None,
                                              log_level=None, start_time=None,
                                              end_time=None, **base_config)
```

Bases: *robot.model.configurer.SuiteConfigurer*

Result suite configured.

Calls suite's `remove_keywords()` and `filter_messages()` 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 suites.

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

**add\_tags**

**end\_body\_item** (*item*)

Called, by default, when keywords, messages or control structures end.

More specific `end_keyword()`, `end_message()`, `:meth:'end_for'`, etc. can be implemented to visit only keywords, messages or specific control structures.

Default implementation does nothing.

**end\_break** (*break\_*)

Called when a BREAK element ends.

By default, calls `end_body_item()` which, by default, does nothing.



**end\_continue** (*continue\_*)  
Called when a CONTINUE element ends.  
By default, calls `end_body_item()` which, by default, does nothing.

**end\_for** (*for\_*)  
Called when a FOR loop ends.  
By default, calls `end_body_item()` which, by default, does nothing.

**end\_for\_iteration** (*iteration*)  
Called when a FOR loop iteration ends.  
By default, calls `end_body_item()` which, by default, does nothing.

**end\_if** (*if\_*)  
Called when an IF/ELSE structure ends.  
By default, calls `end_body_item()` which, by default, does nothing.

**end\_if\_branch** (*branch*)  
Called when an IF/ELSE branch ends.  
By default, calls `end_body_item()` which, by default, does nothing.

**end\_keyword** (*keyword*)  
Called when a keyword ends.  
By default, calls `end_body_item()` which, by default, does nothing.

**end\_message** (*msg*)  
Called when a message ends.  
By default, calls `end_body_item()` which, by default, does nothing.

**end\_return** (*return\_*)  
Called when a RETURN element ends.  
By default, calls `end_body_item()` which, by default, does nothing.

**end\_suite** (*suite*)  
Called when a suite ends. Default implementation does nothing.

**end\_test** (*test*)  
Called when a test ends. Default implementation does nothing.

**end\_try** (*try\_*)  
Called when a TRY/EXCEPT structure ends.  
By default, calls `end_body_item()` which, by default, does nothing.

**end\_try\_branch** (*branch*)  
Called when TRY, EXCEPT, ELSE and FINALLY branches end.  
By default, calls `end_body_item()` which, by default, does nothing.

**end\_while** (*while\_*)  
Called when a WHILE loop ends.  
By default, calls `end_body_item()` which, by default, does nothing.

**end\_while\_iteration** (*iteration*)  
Called when a WHILE loop iteration ends.  
By default, calls `end_body_item()` which, by default, does nothing.

**remove\_tags**

**start\_body\_item** (*item*)

Called, by default, when keywords, messages or control structures start.

More specific `start_keyword()`, `start_message()`, `:meth:'start_for`, etc. can be implemented to visit only keywords, messages or specific control structures.

Can return explicit `False` to stop visiting. Default implementation does nothing.

**start\_break** (*break\_*)

Called when a BREAK element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_continue** (*continue\_*)

Called when a CONTINUE element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_for** (*for\_*)

Called when a FOR loop starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_for\_iteration** (*iteration*)

Called when a FOR loop iteration starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_if** (*if\_*)

Called when an IF/ELSE structure starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_if\_branch** (*branch*)

Called when an IF/ELSE branch starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_keyword** (*keyword*)

Called when a keyword starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_message** (*msg*)

Called when a message starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_return** (*return\_*)

Called when a RETURN element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_suite** (*suite*)

Called when a suite starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

**start\_test** (*test*)

Called when a test starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

**start\_try** (*try\_*)

Called when a TRY/EXCEPT structure starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_try\_branch** (*branch*)

Called when TRY, EXCEPT, ELSE or FINALLY branches start.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_while** (*while\_*)

Called when a WHILE loop starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_while\_iteration** (*iteration*)

Called when a WHILE loop iteration starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**visit\_break** (*break\_*)

Visits BREAK elements.

**visit\_continue** (*continue\_*)

Visits CONTINUE elements.

**visit\_for** (*for\_*)

Implements traversing through FOR loops.

Can be overridden to allow modifying the passed in `for_` without calling `start_for()` or `end_for()` nor visiting body.

**visit\_for\_iteration** (*iteration*)

Implements traversing through single FOR loop iteration.

This is only used with the result side model because on the running side there are no iterations.

Can be overridden to allow modifying the passed in `iteration` without calling `start_for_iteration()` or `end_for_iteration()` nor visiting body.

**visit\_if** (*if\_*)

Implements traversing through IF/ELSE structures.

Notice that `if_` does not have any data directly. Actual IF/ELSE branches are in its `body` and visited using `visit_if_branch()`.

Can be overridden to allow modifying the passed in `if_` without calling `start_if()` or `end_if()` nor visiting branches.

**visit\_if\_branch** (*branch*)

Implements traversing through single IF/ELSE branch.

Can be overridden to allow modifying the passed in `branch` without calling `start_if_branch()` or `end_if_branch()` nor visiting body.

**visit\_keyword** (*kw*)

Implements traversing through keywords.

Can be overridden to allow modifying the passed in `kw` without calling `start_keyword()` or `end_keyword()` nor visiting the body of the keyword

**visit\_message** (*msg*)

Implements visiting messages.

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

**visit\_return** (*return\_*)

Visits a RETURN elements.

**visit\_test** (*test*)

Implements traversing through tests.

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

**visit\_try** (*try\_*)

Implements traversing through TRY/EXCEPT structures.

This method is used with the TRY/EXCEPT root element. Actual TRY, EXCEPT, ELSE and FINALLY branches are visited separately using `visit_try_branch()`.

**visit\_try\_branch** (*branch*)

Visits individual TRY, EXCEPT, ELSE and FINALLY branches.

**visit\_while** (*while\_*)

Implements traversing through WHILE loops.

Can be overridden to allow modifying the passed in `while_` without calling `start_while()` or `end_while()` nor visiting body.

**visit\_while\_iteration** (*iteration*)

Implements traversing through single WHILE loop iteration.

This is only used with the result side model because on the running side there are no iterations.

Can be overridden to allow modifying the passed in `iteration` without calling `start_while_iteration()` or `end_while_iteration()` nor visiting body.

## 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.

**id** = 'errors'

**messages**

A *list-like object* of *Message* instances.

**add** (*other*)

**visit** (*visitor*)

**robot.result.executionresult module**

**class** `robot.result.executionresult.Result` (*source=None, root\_suite=None, errors=None, rpa=None*)

Bases: `object`

Test execution results.

Can be created based on XML output files using the `ExecutionResult()` factory method. Also returned by the `robot.running.TestSuite.run` method.

**source = None**

Path to the XML file where results are read from.

**suite = None**

Hierarchical execution results as a *TestSuite* object.

**errors = None**

Execution errors as an *ExecutionErrors* object.

**statistics**

Test execution statistics.

Statistics are an instance of *Statistics* that is created based on the contained *suite* and possible *configuration*.

Statistics are created every time this property is accessed. Saving them to a variable is thus often a good idea to avoid re-creating them unnecessarily:

```
from robot.api import ExecutionResult

result = ExecutionResult('output.xml')
result.configure(stat_config={'suite_stat_level': 2,
                             'tag_stat_combine': 'tagANDanother'})

stats = result.statistics
print(stats.total.failed)
print(stats.total.passed)
print(stats.tags.combined[0].total)
```

**return\_code**

Return code (integer) of test execution.

By default returns the number of failed 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 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.failed)
print(stats.total.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(tag)

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 keywords.

        Can be overridden to allow modifying the passed in kw without calling start_keyword() or end_keyword() nor visiting the body of the keyword

    visit_for(for_)
        Implements traversing through FOR loops.

        Can be overridden to allow modifying the passed in for_ without calling start_for() or end_for() nor visiting body.
```

**visit\_if\_branch** (*branch*)

Implements traversing through single IF/ELSE branch.

Can be overridden to allow modifying the passed in *branch* without calling `start_if_branch()` or `end_if_branch()` nor visiting body.

**end\_body\_item** (*item*)

Called, by default, when keywords, messages or control structures end.

More specific `end_keyword()`, `end_message()`, `:meth:'end_for`, etc. can be implemented to visit only keywords, messages or specific control structures.

Default implementation does nothing.

**end\_break** (*break\_*)

Called when a BREAK element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_continue** (*continue\_*)

Called when a CONTINUE element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_for** (*for\_*)

Called when a FOR loop ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_for\_iteration** (*iteration*)

Called when a FOR loop iteration ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_if** (*if\_*)

Called when an IF/ELSE structure ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_if\_branch** (*branch*)

Called when an IF/ELSE branch ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_keyword** (*keyword*)

Called when a keyword ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_message** (*msg*)

Called when a message ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_return** (*return\_*)

Called when a RETURN element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_suite** (*suite*)

Called when a suite ends. Default implementation does nothing.

**end\_test** (*test*)

Called when a test ends. Default implementation does nothing.



**end\_try** (*try\_*)

Called when a TRY/EXCEPT structure ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_try\_branch** (*branch*)

Called when TRY, EXCEPT, ELSE and FINALLY branches end.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_while** (*while\_*)

Called when a WHILE loop ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_while\_iteration** (*iteration*)

Called when a WHILE loop iteration ends.

By default, calls `end_body_item()` which, by default, does nothing.

**start\_body\_item** (*item*)

Called, by default, when keywords, messages or control structures start.

More specific `start_keyword()`, `start_message()`, `:meth:'start_for`, etc. can be implemented to visit only keywords, messages or specific control structures.

Can return explicit `False` to stop visiting. Default implementation does nothing.

**start\_break** (*break\_*)

Called when a BREAK element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_continue** (*continue\_*)

Called when a CONTINUE element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_for** (*for\_*)

Called when a FOR loop starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_for\_iteration** (*iteration*)

Called when a FOR loop iteration starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_if** (*if\_*)

Called when an IF/ELSE structure starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_if\_branch** (*branch*)

Called when an IF/ELSE branch starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_keyword** (*keyword*)

Called when a keyword starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_message** (*msg*)

Called when a message starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_return** (*return\_*)

Called when a RETURN element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_suite** (*suite*)

Called when a suite starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

**start\_test** (*test*)

Called when a test starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

**start\_try** (*try\_*)

Called when a TRY/EXCEPT structure starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_try\_branch** (*branch*)

Called when TRY, EXCEPT, ELSE or FINALLY branches start.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_while** (*while\_*)

Called when a WHILE loop starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_while\_iteration** (*iteration*)

Called when a WHILE loop iteration starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**visit\_break** (*break\_*)

Visits BREAK elements.

**visit\_continue** (*continue\_*)

Visits CONTINUE elements.

**visit\_for\_iteration** (*iteration*)

Implements traversing through single FOR loop iteration.

This is only used with the result side model because on the running side there are no iterations.

Can be overridden to allow modifying the passed in *iteration* without calling *start\_for\_iteration()* or *end\_for\_iteration()* nor visiting body.

**visit\_if** (*if\_*)

Implements traversing through IF/ELSE structures.

Notice that *if\_* does not have any data directly. Actual IF/ELSE branches are in its body and visited using *visit\_if\_branch()*.

Can be overridden to allow modifying the passed in *if\_* without calling *start\_if()* or *end\_if()* nor visiting branches.

**visit\_message** (*msg*)

Implements visiting messages.

Can be overridden to allow modifying the passed in *msg* without calling *start\_message()* or *end\_message()*.

**visit\_return** (*return\_*)

Visits a RETURN elements.

**visit\_suite** (*suite*)

Implements traversing through suites.

Can be overridden to allow modifying the passed in *suite* without calling *start\_suite()* or *end\_suite()* nor visiting child suites, tests or setup and teardown at all.

**visit\_test** (*test*)

Implements traversing through tests.

Can be overridden to allow modifying the passed in *test* without calling *start\_test()* or *end\_test()* nor visiting the body of the test.

**visit\_try** (*try\_*)

Implements traversing through TRY/EXCEPT structures.

This method is used with the TRY/EXCEPT root element. Actual TRY, EXCEPT, ELSE and FINALLY branches are visited separately using *visit\_try\_branch()*.

**visit\_try\_branch** (*branch*)

Visits individual TRY, EXCEPT, ELSE and FINALLY branches.

**visit\_while** (*while\_*)

Implements traversing through WHILE loops.

Can be overridden to allow modifying the passed in *while\_* without calling *start\_while()* or *end\_while()* nor visiting body.

**visit\_while\_iteration** (*iteration*)

Implements traversing through single WHILE loop iteration.

This is only used with the result side model because on the running side there are no iterations.

Can be overridden to allow modifying the passed in *iteration* without calling *start\_while\_iteration()* or *end\_while\_iteration()* nor visiting body.

**class** robot.result.keywordremover.PassedKeywordRemover

Bases: robot.result.keywordremover.\_KeywordRemover

**start\_suite** (*suite*)

Called when a suite starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

**visit\_test** (*test*)

Implements traversing through tests.

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

**visit\_keyword** (*keyword*)

Implements traversing through keywords.

Can be overridden to allow modifying the passed in `kw` without calling `start_keyword()` or `end_keyword()` nor visiting the body of the keyword

**end\_body\_item** (*item*)

Called, by default, when keywords, messages or control structures end.

More specific `end_keyword()`, `end_message()`, `:meth:'end_for`, etc. can be implemented to visit only keywords, messages or specific control structures.

Default implementation does nothing.

**end\_break** (*break\_*)

Called when a BREAK element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_continue** (*continue\_*)

Called when a CONTINUE element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_for** (*for\_*)

Called when a FOR loop ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_for\_iteration** (*iteration*)

Called when a FOR loop iteration ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_if** (*if\_*)

Called when an IF/ELSE structure ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_if\_branch** (*branch*)

Called when an IF/ELSE branch ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_keyword** (*keyword*)

Called when a keyword ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_message** (*msg*)

Called when a message ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_return** (*return\_*)

Called when a RETURN element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_suite** (*suite*)

Called when a suite ends. Default implementation does nothing.

**end\_test** (*test*)

Called when a test ends. Default implementation does nothing.

**end\_try** (*try\_*)

Called when a TRY/EXCEPT structure ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_try\_branch** (*branch*)

Called when TRY, EXCEPT, ELSE and FINALLY branches end.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_while** (*while\_*)

Called when a WHILE loop ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_while\_iteration** (*iteration*)

Called when a WHILE loop iteration ends.

By default, calls `end_body_item()` which, by default, does nothing.

**start\_body\_item** (*item*)

Called, by default, when keywords, messages or control structures start.

More specific `start_keyword()`, `start_message()`, `:meth:'start_for`, etc. can be implemented to visit only keywords, messages or specific control structures.

Can return explicit `False` to stop visiting. Default implementation does nothing.

**start\_break** (*break\_*)

Called when a BREAK element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_continue** (*continue\_*)

Called when a CONTINUE element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_for** (*for\_*)

Called when a FOR loop starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_for\_iteration** (*iteration*)

Called when a FOR loop iteration starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_if** (*if\_*)

Called when an IF/ELSE structure starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_if\_branch** (*branch*)

Called when an IF/ELSE branch starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_keyword** (*keyword*)

Called when a keyword starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_message** (*msg*)

Called when a message starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_return** (*return\_*)

Called when a RETURN element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_test** (*test*)

Called when a test starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

**start\_try** (*try\_*)

Called when a TRY/EXCEPT structure starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_try\_branch** (*branch*)

Called when TRY, EXCEPT, ELSE or FINALLY branches start.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_while** (*while\_*)

Called when a WHILE loop starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_while\_iteration** (*iteration*)

Called when a WHILE loop iteration starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**visit\_break** (*break\_*)

Visits BREAK elements.

**visit\_continue** (*continue\_*)

Visits CONTINUE elements.

**visit\_for** (*for\_*)

Implements traversing through FOR loops.

Can be overridden to allow modifying the passed in *for\_* without calling *start\_for()* or *end\_for()* nor visiting body.

**visit\_for\_iteration** (*iteration*)

Implements traversing through single FOR loop iteration.

This is only used with the result side model because on the running side there are no iterations.

Can be overridden to allow modifying the passed in *iteration* without calling *start\_for\_iteration()* or *end\_for\_iteration()* nor visiting body.

**visit\_if** (*if\_*)

Implements traversing through IF/ELSE structures.

Notice that *if\_* does not have any data directly. Actual IF/ELSE branches are in its body and visited using *visit\_if\_branch()*.

Can be overridden to allow modifying the passed in *if\_* without calling *start\_if()* or *end\_if()* nor visiting branches.

**visit\_if\_branch** (*branch*)

Implements traversing through single IF/ELSE branch.

Can be overridden to allow modifying the passed in *branch* without calling *start\_if\_branch()* or *end\_if\_branch()* nor visiting body.

**visit\_message** (*msg*)

Implements visiting messages.

Can be overridden to allow modifying the passed in *msg* without calling *start\_message()* or *end\_message()*.

**visit\_return** (*return\_*)

Visits a RETURN elements.

**visit\_suite** (*suite*)

Implements traversing through suites.

Can be overridden to allow modifying the passed in *suite* without calling *start\_suite()* or *end\_suite()* nor visiting child suites, tests or setup and teardown at all.

**visit\_try** (*try\_*)

Implements traversing through TRY/EXCEPT structures.

This method is used with the TRY/EXCEPT root element. Actual TRY, EXCEPT, ELSE and FINALLY branches are visited separately using *visit\_try\_branch()*.

**visit\_try\_branch** (*branch*)

Visits individual TRY, EXCEPT, ELSE and FINALLY branches.

**visit\_while** (*while\_*)

Implements traversing through WHILE loops.

Can be overridden to allow modifying the passed in *while\_* without calling *start\_while()* or *end\_while()* nor visiting body.

**visit\_while\_iteration** (*iteration*)

Implements traversing through single WHILE loop iteration.

This is only used with the result side model because on the running side there are no iterations.

Can be overridden to allow modifying the passed in *iteration* without calling `start_while_iteration()` or `end_while_iteration()` nor visiting body.

**class** `robot.result.keywordremover.ByNameKeywordRemover` (*pattern*)

Bases: `robot.result.keywordremover._KeywordRemover`

**start\_keyword** (*kw*)

Called when a keyword starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**end\_body\_item** (*item*)

Called, by default, when keywords, messages or control structures end.

More specific `end_keyword()`, `end_message()`, `:meth:'end_for'`, etc. can be implemented to visit only keywords, messages or specific control structures.

Default implementation does nothing.

**end\_break** (*break\_*)

Called when a BREAK element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_continue** (*continue\_*)

Called when a CONTINUE element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_for** (*for\_*)

Called when a FOR loop ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_for\_iteration** (*iteration*)

Called when a FOR loop iteration ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_if** (*if\_*)

Called when an IF/ELSE structure ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_if\_branch** (*branch*)

Called when an IF/ELSE branch ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_keyword** (*keyword*)

Called when a keyword ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_message** (*msg*)

Called when a message ends.

By default, calls `end_body_item()` which, by default, does nothing.



**end\_return** (*return\_*)

Called when a RETURN element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_suite** (*suite*)

Called when a suite ends. Default implementation does nothing.

**end\_test** (*test*)

Called when a test ends. Default implementation does nothing.

**end\_try** (*try\_*)

Called when a TRY/EXCEPT structure ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_try\_branch** (*branch*)

Called when TRY, EXCEPT, ELSE and FINALLY branches end.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_while** (*while\_*)

Called when a WHILE loop ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_while\_iteration** (*iteration*)

Called when a WHILE loop iteration ends.

By default, calls `end_body_item()` which, by default, does nothing.

**start\_body\_item** (*item*)

Called, by default, when keywords, messages or control structures start.

More specific `start_keyword()`, `start_message()`, `:meth:'start_for`, etc. can be implemented to visit only keywords, messages or specific control structures.

Can return explicit `False` to stop visiting. Default implementation does nothing.

**start\_break** (*break\_*)

Called when a BREAK element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_continue** (*continue\_*)

Called when a CONTINUE element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_for** (*for\_*)

Called when a FOR loop starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_for\_iteration** (*iteration*)

Called when a FOR loop iteration starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_if** (*if\_*)

Called when an IF/ELSE structure starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_if\_branch** (*branch*)

Called when an IF/ELSE branch starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_message** (*msg*)

Called when a message starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_return** (*return\_*)

Called when a RETURN element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_suite** (*suite*)

Called when a suite starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

**start\_test** (*test*)

Called when a test starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

**start\_try** (*try\_*)

Called when a TRY/EXCEPT structure starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_try\_branch** (*branch*)

Called when TRY, EXCEPT, ELSE or FINALLY branches start.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_while** (*while\_*)

Called when a WHILE loop starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_while\_iteration** (*iteration*)

Called when a WHILE loop iteration starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**visit\_break** (*break\_*)

Visits BREAK elements.

**visit\_continue** (*continue\_*)  
Visits CONTINUE elements.

**visit\_for** (*for\_*)  
Implements traversing through FOR loops.

Can be overridden to allow modifying the passed in *for\_* without calling *start\_for()* or *end\_for()* nor visiting body.

**visit\_for\_iteration** (*iteration*)  
Implements traversing through single FOR loop iteration.

This is only used with the result side model because on the running side there are no iterations.

Can be overridden to allow modifying the passed in *iteration* without calling *start\_for\_iteration()* or *end\_for\_iteration()* nor visiting body.

**visit\_if** (*if\_*)  
Implements traversing through IF/ELSE structures.

Notice that *if\_* does not have any data directly. Actual IF/ELSE branches are in its body and visited using *visit\_if\_branch()*.

Can be overridden to allow modifying the passed in *if\_* without calling *start\_if()* or *end\_if()* nor visiting branches.

**visit\_if\_branch** (*branch*)  
Implements traversing through single IF/ELSE branch.

Can be overridden to allow modifying the passed in *branch* without calling *start\_if\_branch()* or *end\_if\_branch()* nor visiting body.

**visit\_keyword** (*kw*)  
Implements traversing through keywords.

Can be overridden to allow modifying the passed in *kw* without calling *start\_keyword()* or *end\_keyword()* nor visiting the body of the keyword

**visit\_message** (*msg*)  
Implements visiting messages.

Can be overridden to allow modifying the passed in *msg* without calling *start\_message()* or *end\_message()*.

**visit\_return** (*return\_*)  
Visits a RETURN elements.

**visit\_suite** (*suite*)  
Implements traversing through suites.

Can be overridden to allow modifying the passed in *suite* without calling *start\_suite()* or *end\_suite()* nor visiting child suites, tests or setup and teardown at all.

**visit\_test** (*test*)  
Implements traversing through tests.

Can be overridden to allow modifying the passed in *test* without calling *start\_test()* or *end\_test()* nor visiting the body of the test.

**visit\_try** (*try\_*)  
Implements traversing through TRY/EXCEPT structures.

This method is used with the TRY/EXCEPT root element. Actual TRY, EXCEPT, ELSE and FINALLY branches are visited separately using *visit\_try\_branch()*.

**visit\_try\_branch** (*branch*)

Visits individual TRY, EXCEPT, ELSE and FINALLY branches.

**visit\_while** (*while\_*)

Implements traversing through WHILE loops.

Can be overridden to allow modifying the passed in *while\_* without calling *start\_while()* or *end\_while()* nor visiting body.

**visit\_while\_iteration** (*iteration*)

Implements traversing through single WHILE loop iteration.

This is only used with the result side model because on the running side there are no iterations.

Can be overridden to allow modifying the passed in *iteration* without calling *start\_while\_iteration()* or *end\_while\_iteration()* nor visiting body.

**class** robot.result.keywordremover.**ByTagKeywordRemover** (*pattern*)

Bases: robot.result.keywordremover.\_KeywordRemover

**start\_keyword** (*kw*)

Called when a keyword starts.

By default, calls *start\_body\_item()* which, by default, does nothing.

Can return explicit False to stop visiting.

**end\_body\_item** (*item*)

Called, by default, when keywords, messages or control structures end.

More specific *end\_keyword()*, *end\_message()*, *:meth:'end\_for'*, etc. can be implemented to visit only keywords, messages or specific control structures.

Default implementation does nothing.

**end\_break** (*break\_*)

Called when a BREAK element ends.

By default, calls *end\_body\_item()* which, by default, does nothing.

**end\_continue** (*continue\_*)

Called when a CONTINUE element ends.

By default, calls *end\_body\_item()* which, by default, does nothing.

**end\_for** (*for\_*)

Called when a FOR loop ends.

By default, calls *end\_body\_item()* which, by default, does nothing.

**end\_for\_iteration** (*iteration*)

Called when a FOR loop iteration ends.

By default, calls *end\_body\_item()* which, by default, does nothing.

**end\_if** (*if\_*)

Called when an IF/ELSE structure ends.

By default, calls *end\_body\_item()* which, by default, does nothing.

**end\_if\_branch** (*branch*)

Called when an IF/ELSE branch ends.

By default, calls *end\_body\_item()* which, by default, does nothing.

**end\_keyword** (*keyword*)

Called when a keyword ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_message** (*msg*)

Called when a message ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_return** (*return\_*)

Called when a RETURN element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_suite** (*suite*)

Called when a suite ends. Default implementation does nothing.

**end\_test** (*test*)

Called when a test ends. Default implementation does nothing.

**end\_try** (*try\_*)

Called when a TRY/EXCEPT structure ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_try\_branch** (*branch*)

Called when TRY, EXCEPT, ELSE and FINALLY branches end.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_while** (*while\_*)

Called when a WHILE loop ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_while\_iteration** (*iteration*)

Called when a WHILE loop iteration ends.

By default, calls `end_body_item()` which, by default, does nothing.

**start\_body\_item** (*item*)

Called, by default, when keywords, messages or control structures start.

More specific `start_keyword()`, `start_message()`, `:meth:'start_for`, etc. can be implemented to visit only keywords, messages or specific control structures.

Can return explicit `False` to stop visiting. Default implementation does nothing.

**start\_break** (*break\_*)

Called when a BREAK element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_continue** (*continue\_*)

Called when a CONTINUE element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_for** (*for\_*)

Called when a FOR loop starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_for\_iteration** (*iteration*)

Called when a FOR loop iteration starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_if** (*if\_*)

Called when an IF/ELSE structure starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_if\_branch** (*branch*)

Called when an IF/ELSE branch starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_message** (*msg*)

Called when a message starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_return** (*return\_*)

Called when a RETURN element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_suite** (*suite*)

Called when a suite starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

**start\_test** (*test*)

Called when a test starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

**start\_try** (*try\_*)

Called when a TRY/EXCEPT structure starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_try\_branch** (*branch*)

Called when TRY, EXCEPT, ELSE or FINALLY branches start.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_while** (*while\_*)

Called when a WHILE loop starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_while\_iteration** (*iteration*)

Called when a WHILE loop iteration starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**visit\_break** (*break\_*)

Visits BREAK elements.

**visit\_continue** (*continue\_*)

Visits CONTINUE elements.

**visit\_for** (*for\_*)

Implements traversing through FOR loops.

Can be overridden to allow modifying the passed in `for_` without calling `start_for()` or `end_for()` nor visiting body.

**visit\_for\_iteration** (*iteration*)

Implements traversing through single FOR loop iteration.

This is only used with the result side model because on the running side there are no iterations.

Can be overridden to allow modifying the passed in `iteration` without calling `start_for_iteration()` or `end_for_iteration()` nor visiting body.

**visit\_if** (*if\_*)

Implements traversing through IF/ELSE structures.

Notice that `if_` does not have any data directly. Actual IF/ELSE branches are in its body and visited using `visit_if_branch()`.

Can be overridden to allow modifying the passed in `if_` without calling `start_if()` or `end_if()` nor visiting branches.

**visit\_if\_branch** (*branch*)

Implements traversing through single IF/ELSE branch.

Can be overridden to allow modifying the passed in `branch` without calling `start_if_branch()` or `end_if_branch()` nor visiting body.

**visit\_keyword** (*kw*)

Implements traversing through keywords.

Can be overridden to allow modifying the passed in `kw` without calling `start_keyword()` or `end_keyword()` nor visiting the body of the keyword

**visit\_message** (*msg*)

Implements visiting messages.

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

**visit\_return** (*return\_*)

Visits a RETURN elements.

**visit\_suite** (*suite*)

Implements traversing through suites.

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

**visit\_test** (*test*)

Implements traversing through tests.

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

**visit\_try** (*try\_*)

Implements traversing through TRY/EXCEPT structures.

This method is used with the TRY/EXCEPT root element. Actual TRY, EXCEPT, ELSE and FINALLY branches are visited separately using `visit_try_branch()`.

**visit\_try\_branch** (*branch*)

Visits individual TRY, EXCEPT, ELSE and FINALLY branches.

**visit\_while** (*while\_*)

Implements traversing through WHILE loops.

Can be overridden to allow modifying the passed in `while_` without calling `start_while()` or `end_while()` nor visiting body.

**visit\_while\_iteration** (*iteration*)

Implements traversing through single WHILE loop iteration.

This is only used with the result side model because on the running side there are no iterations.

Can be overridden to allow modifying the passed in `iteration` without calling `start_while_iteration()` or `end_while_iteration()` nor visiting body.

**class** `robot.result.keywordremover.ForLoopItemsRemover`

Bases: `robot.result.keywordremover._LoopItemsRemover`

**start\_for** (*for\_*)

Called when a FOR loop starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**end\_body\_item** (*item*)

Called, by default, when keywords, messages or control structures end.

More specific `end_keyword()`, `end_message()`, `:meth:'end_for'`, etc. can be implemented to visit only keywords, messages or specific control structures.

Default implementation does nothing.

**end\_break** (*break\_*)

Called when a BREAK element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_continue** (*continue\_*)

Called when a CONTINUE element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_for** (*for\_*)

Called when a FOR loop ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_for\_iteration** (*iteration*)

Called when a FOR loop iteration ends.

By default, calls `end_body_item()` which, by default, does nothing.



**end\_if** (*if\_*)

Called when an IF/ELSE structure ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_if\_branch** (*branch*)

Called when an IF/ELSE branch ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_keyword** (*keyword*)

Called when a keyword ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_message** (*msg*)

Called when a message ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_return** (*return\_*)

Called when a RETURN element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_suite** (*suite*)

Called when a suite ends. Default implementation does nothing.

**end\_test** (*test*)

Called when a test ends. Default implementation does nothing.

**end\_try** (*try\_*)

Called when a TRY/EXCEPT structure ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_try\_branch** (*branch*)

Called when TRY, EXCEPT, ELSE and FINALLY branches end.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_while** (*while\_*)

Called when a WHILE loop ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_while\_iteration** (*iteration*)

Called when a WHILE loop iteration ends.

By default, calls `end_body_item()` which, by default, does nothing.

**start\_body\_item** (*item*)

Called, by default, when keywords, messages or control structures start.

More specific `start_keyword()`, `start_message()`, `:meth:'start_for`, etc. can be implemented to visit only keywords, messages or specific control structures.

Can return explicit `False` to stop visiting. Default implementation does nothing.

**start\_break** (*break\_*)

Called when a BREAK element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_continue** (*continue\_*)

Called when a CONTINUE element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_for\_iteration** (*iteration*)

Called when a FOR loop iteration starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_if** (*if\_*)

Called when an IF/ELSE structure starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_if\_branch** (*branch*)

Called when an IF/ELSE branch starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_keyword** (*keyword*)

Called when a keyword starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_message** (*msg*)

Called when a message starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_return** (*return\_*)

Called when a RETURN element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_suite** (*suite*)

Called when a suite starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

**start\_test** (*test*)

Called when a test starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

**start\_try** (*try\_*)

Called when a TRY/EXCEPT structure starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_try\_branch** (*branch*)

Called when TRY, EXCEPT, ELSE or FINALLY branches start.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_while** (*while\_*)

Called when a WHILE loop starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_while\_iteration** (*iteration*)

Called when a WHILE loop iteration starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**visit\_break** (*break\_*)

Visits BREAK elements.

**visit\_continue** (*continue\_*)

Visits CONTINUE elements.

**visit\_for** (*for\_*)

Implements traversing through FOR loops.

Can be overridden to allow modifying the passed in `for_` without calling `start_for()` or `end_for()` nor visiting body.

**visit\_for\_iteration** (*iteration*)

Implements traversing through single FOR loop iteration.

This is only used with the result side model because on the running side there are no iterations.

Can be overridden to allow modifying the passed in `iteration` without calling `start_for_iteration()` or `end_for_iteration()` nor visiting body.

**visit\_if** (*if\_*)

Implements traversing through IF/ELSE structures.

Notice that `if_` does not have any data directly. Actual IF/ELSE branches are in its body and visited using `visit_if_branch()`.

Can be overridden to allow modifying the passed in `if_` without calling `start_if()` or `end_if()` nor visiting branches.

**visit\_if\_branch** (*branch*)

Implements traversing through single IF/ELSE branch.

Can be overridden to allow modifying the passed in `branch` without calling `start_if_branch()` or `end_if_branch()` nor visiting body.

**visit\_keyword** (*kw*)

Implements traversing through keywords.

Can be overridden to allow modifying the passed in `kw` without calling `start_keyword()` or `end_keyword()` nor visiting the body of the keyword

**visit\_message** (*msg*)

Implements visiting messages.

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

**visit\_return** (*return\_*)  
Visits a RETURN elements.

**visit\_suite** (*suite*)  
Implements traversing through suites.

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

**visit\_test** (*test*)  
Implements traversing through tests.

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

**visit\_try** (*try\_*)  
Implements traversing through TRY/EXCEPT structures.

This method is used with the TRY/EXCEPT root element. Actual TRY, EXCEPT, ELSE and FINALLY branches are visited separately using `visit_try_branch()`.

**visit\_try\_branch** (*branch*)  
Visits individual TRY, EXCEPT, ELSE and FINALLY branches.

**visit\_while** (*while\_*)  
Implements traversing through WHILE loops.

Can be overridden to allow modifying the passed in `while_` without calling `start_while()` or `end_while()` nor visiting body.

**visit\_while\_iteration** (*iteration*)  
Implements traversing through single WHILE loop iteration.

This is only used with the result side model because on the running side there are no iterations.

Can be overridden to allow modifying the passed in `iteration` without calling `start_while_iteration()` or `end_while_iteration()` nor visiting body.

**class** `robot.result.keywordremover.WhileLoopItemsRemover`  
Bases: `robot.result.keywordremover._LoopItemsRemover`

**start\_while** (*while\_*)  
Called when a WHILE loop starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**end\_body\_item** (*item*)  
Called, by default, when keywords, messages or control structures end.

More specific `end_keyword()`, `end_message()`, `:meth:'end_for'`, etc. can be implemented to visit only keywords, messages or specific control structures.

Default implementation does nothing.

**end\_break** (*break\_*)  
Called when a BREAK element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_continue** (*continue\_*)  
Called when a CONTINUE element ends.  
By default, calls `end_body_item()` which, by default, does nothing.

**end\_for** (*for\_*)  
Called when a FOR loop ends.  
By default, calls `end_body_item()` which, by default, does nothing.

**end\_for\_iteration** (*iteration*)  
Called when a FOR loop iteration ends.  
By default, calls `end_body_item()` which, by default, does nothing.

**end\_if** (*if\_*)  
Called when an IF/ELSE structure ends.  
By default, calls `end_body_item()` which, by default, does nothing.

**end\_if\_branch** (*branch*)  
Called when an IF/ELSE branch ends.  
By default, calls `end_body_item()` which, by default, does nothing.

**end\_keyword** (*keyword*)  
Called when a keyword ends.  
By default, calls `end_body_item()` which, by default, does nothing.

**end\_message** (*msg*)  
Called when a message ends.  
By default, calls `end_body_item()` which, by default, does nothing.

**end\_return** (*return\_*)  
Called when a RETURN element ends.  
By default, calls `end_body_item()` which, by default, does nothing.

**end\_suite** (*suite*)  
Called when a suite ends. Default implementation does nothing.

**end\_test** (*test*)  
Called when a test ends. Default implementation does nothing.

**end\_try** (*try\_*)  
Called when a TRY/EXCEPT structure ends.  
By default, calls `end_body_item()` which, by default, does nothing.

**end\_try\_branch** (*branch*)  
Called when TRY, EXCEPT, ELSE and FINALLY branches end.  
By default, calls `end_body_item()` which, by default, does nothing.

**end\_while** (*while\_*)  
Called when a WHILE loop ends.  
By default, calls `end_body_item()` which, by default, does nothing.

**end\_while\_iteration** (*iteration*)  
Called when a WHILE loop iteration ends.  
By default, calls `end_body_item()` which, by default, does nothing.

**start\_body\_item** (*item*)

Called, by default, when keywords, messages or control structures start.

More specific `start_keyword()`, `start_message()`, `:meth:'start_for`, etc. can be implemented to visit only keywords, messages or specific control structures.

Can return explicit `False` to stop visiting. Default implementation does nothing.

**start\_break** (*break\_*)

Called when a BREAK element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_continue** (*continue\_*)

Called when a CONTINUE element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_for** (*for\_*)

Called when a FOR loop starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_for\_iteration** (*iteration*)

Called when a FOR loop iteration starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_if** (*if\_*)

Called when an IF/ELSE structure starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_if\_branch** (*branch*)

Called when an IF/ELSE branch starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_keyword** (*keyword*)

Called when a keyword starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_message** (*msg*)

Called when a message starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_return** (*return\_*)

Called when a RETURN element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_suite** (*suite*)

Called when a suite starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

**start\_test** (*test*)

Called when a test starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

**start\_try** (*try\_*)

Called when a TRY/EXCEPT structure starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_try\_branch** (*branch*)

Called when TRY, EXCEPT, ELSE or FINALLY branches start.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_while\_iteration** (*iteration*)

Called when a WHILE loop iteration starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**visit\_break** (*break\_*)

Visits BREAK elements.

**visit\_continue** (*continue\_*)

Visits CONTINUE elements.

**visit\_for** (*for\_*)

Implements traversing through FOR loops.

Can be overridden to allow modifying the passed in `for_` without calling `start_for()` or `end_for()` nor visiting body.

**visit\_for\_iteration** (*iteration*)

Implements traversing through single FOR loop iteration.

This is only used with the result side model because on the running side there are no iterations.

Can be overridden to allow modifying the passed in `iteration` without calling `start_for_iteration()` or `end_for_iteration()` nor visiting body.

**visit\_if** (*if\_*)

Implements traversing through IF/ELSE structures.

Notice that `if_` does not have any data directly. Actual IF/ELSE branches are in its `body` and visited using `visit_if_branch()`.

Can be overridden to allow modifying the passed in `if_` without calling `start_if()` or `end_if()` nor visiting branches.

**visit\_if\_branch** (*branch*)

Implements traversing through single IF/ELSE branch.

Can be overridden to allow modifying the passed in `branch` without calling `start_if_branch()` or `end_if_branch()` nor visiting body.

**visit\_keyword** (*kw*)

Implements traversing through keywords.

Can be overridden to allow modifying the passed in `kw` without calling `start_keyword()` or `end_keyword()` nor visiting the body of the keyword

**visit\_message** (*msg*)

Implements visiting messages.

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

**visit\_return** (*return\_*)

Visits a RETURN elements.

**visit\_suite** (*suite*)

Implements traversing through suites.

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

**visit\_test** (*test*)

Implements traversing through tests.

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

**visit\_try** (*try\_*)

Implements traversing through TRY/EXCEPT structures.

This method is used with the TRY/EXCEPT root element. Actual TRY, EXCEPT, ELSE and FINALLY branches are visited separately using `visit_try_branch()`.

**visit\_try\_branch** (*branch*)

Visits individual TRY, EXCEPT, ELSE and FINALLY branches.

**visit\_while** (*while\_*)

Implements traversing through WHILE loops.

Can be overridden to allow modifying the passed in `while_` without calling `start_while()` or `end_while()` nor visiting body.

**visit\_while\_iteration** (*iteration*)

Implements traversing through single WHILE loop iteration.

This is only used with the result side model because on the running side there are no iterations.

Can be overridden to allow modifying the passed in `iteration` without calling `start_while_iteration()` or `end_while_iteration()` nor visiting body.

**class** robot.result.keywordremover.WaitUntilKeywordSucceedsRemover

Bases: robot.result.keywordremover.\_KeywordRemover

**start\_keyword** (*kw*)

Called when a keyword starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.



**end\_body\_item** (*item*)

Called, by default, when keywords, messages or control structures end.

More specific `end_keyword()`, `end_message()`, `:meth:'end_for'`, etc. can be implemented to visit only keywords, messages or specific control structures.

Default implementation does nothing.

**end\_break** (*break\_*)

Called when a BREAK element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_continue** (*continue\_*)

Called when a CONTINUE element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_for** (*for\_*)

Called when a FOR loop ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_for\_iteration** (*iteration*)

Called when a FOR loop iteration ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_if** (*if\_*)

Called when an IF/ELSE structure ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_if\_branch** (*branch*)

Called when an IF/ELSE branch ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_keyword** (*keyword*)

Called when a keyword ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_message** (*msg*)

Called when a message ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_return** (*return\_*)

Called when a RETURN element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_suite** (*suite*)

Called when a suite ends. Default implementation does nothing.

**end\_test** (*test*)

Called when a test ends. Default implementation does nothing.

**end\_try** (*try\_*)

Called when a TRY/EXCEPT structure ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_try\_branch** (*branch*)

Called when TRY, EXCEPT, ELSE and FINALLY branches end.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_while** (*while\_*)

Called when a WHILE loop ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_while\_iteration** (*iteration*)

Called when a WHILE loop iteration ends.

By default, calls `end_body_item()` which, by default, does nothing.

**start\_body\_item** (*item*)

Called, by default, when keywords, messages or control structures start.

More specific `start_keyword()`, `start_message()`, `:meth:'start_for`, etc. can be implemented to visit only keywords, messages or specific control structures.

Can return explicit `False` to stop visiting. Default implementation does nothing.

**start\_break** (*break\_*)

Called when a BREAK element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_continue** (*continue\_*)

Called when a CONTINUE element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_for** (*for\_*)

Called when a FOR loop starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_for\_iteration** (*iteration*)

Called when a FOR loop iteration starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_if** (*if\_*)

Called when an IF/ELSE structure starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_if\_branch** (*branch*)

Called when an IF/ELSE branch starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_message** (*msg*)

Called when a message starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_return** (*return\_*)

Called when a RETURN element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_suite** (*suite*)

Called when a suite starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

**start\_test** (*test*)

Called when a test starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

**start\_try** (*try\_*)

Called when a TRY/EXCEPT structure starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_try\_branch** (*branch*)

Called when TRY, EXCEPT, ELSE or FINALLY branches start.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_while** (*while\_*)

Called when a WHILE loop starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_while\_iteration** (*iteration*)

Called when a WHILE loop iteration starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**visit\_break** (*break\_*)

Visits BREAK elements.

**visit\_continue** (*continue\_*)

Visits CONTINUE elements.

**visit\_for** (*for\_*)

Implements traversing through FOR loops.

Can be overridden to allow modifying the passed in `for_` without calling `start_for()` or `end_for()` nor visiting body.

**visit\_for\_iteration** (*iteration*)

Implements traversing through single FOR loop iteration.

This is only used with the result side model because on the running side there are no iterations.

Can be overridden to allow modifying the passed in `iteration` without calling `start_for_iteration()` or `end_for_iteration()` nor visiting body.

**visit\_if** (*if\_*)

Implements traversing through IF/ELSE structures.

Notice that `if_` does not have any data directly. Actual IF/ELSE branches are in its body and visited using `visit_if_branch()`.

Can be overridden to allow modifying the passed in `if_` without calling `start_if()` or `end_if()` nor visiting branches.

**visit\_if\_branch** (*branch*)

Implements traversing through single IF/ELSE branch.

Can be overridden to allow modifying the passed in `branch` without calling `start_if_branch()` or `end_if_branch()` nor visiting body.

**visit\_keyword** (*kw*)

Implements traversing through keywords.

Can be overridden to allow modifying the passed in `kw` without calling `start_keyword()` or `end_keyword()` nor visiting the body of the keyword

**visit\_message** (*msg*)

Implements visiting messages.

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

**visit\_return** (*return\_*)

Visits a RETURN elements.

**visit\_suite** (*suite*)

Implements traversing through suites.

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

**visit\_test** (*test*)

Implements traversing through tests.

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

**visit\_try** (*try\_*)

Implements traversing through TRY/EXCEPT structures.

This method is used with the TRY/EXCEPT root element. Actual TRY, EXCEPT, ELSE and FINALLY branches are visited separately using `visit_try_branch()`.

**visit\_try\_branch** (*branch*)

Visits individual TRY, EXCEPT, ELSE and FINALLY branches.

**visit\_while** (*while\_*)

Implements traversing through WHILE loops.

Can be overridden to allow modifying the passed in `while_` without calling `start_while()` or `end_while()` nor visiting body.

**visit\_while\_iteration** (*iteration*)

Implements traversing through single WHILE loop iteration.

This is only used with the result side model because on the running side there are no iterations.

Can be overridden to allow modifying the passed in `iteration` without calling `start_while_iteration()` or `end_while_iteration()` nor visiting body.

**class** `robot.result.keywordremover.WarningAndErrorFinder`

Bases: `robot.model.visitor.SuiteVisitor`

**start\_suite** (*suite*)

Called when a suite starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

**start\_test** (*test*)

Called when a test starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

**start\_keyword** (*keyword*)

Called when a keyword starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**visit\_message** (*msg*)

Implements visiting messages.

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

**end\_body\_item** (*item*)

Called, by default, when keywords, messages or control structures end.

More specific `end_keyword()`, `end_message()`, `:meth:'end_for'`, etc. can be implemented to visit only keywords, messages or specific control structures.

Default implementation does nothing.

**end\_break** (*break\_*)

Called when a BREAK element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_continue** (*continue\_*)

Called when a CONTINUE element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_for** (*for\_*)

Called when a FOR loop ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_for\_iteration** (*iteration*)

Called when a FOR loop iteration ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_if** (*if\_*)

Called when an IF/ELSE structure ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_if\_branch** (*branch*)

Called when an IF/ELSE branch ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_keyword** (*keyword*)

Called when a keyword ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_message** (*msg*)

Called when a message ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_return** (*return\_*)

Called when a RETURN element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_suite** (*suite*)

Called when a suite ends. Default implementation does nothing.

**end\_test** (*test*)

Called when a test ends. Default implementation does nothing.

**end\_try** (*try\_*)

Called when a TRY/EXCEPT structure ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_try\_branch** (*branch*)

Called when TRY, EXCEPT, ELSE and FINALLY branches end.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_while** (*while\_*)

Called when a WHILE loop ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_while\_iteration** (*iteration*)

Called when a WHILE loop iteration ends.

By default, calls `end_body_item()` which, by default, does nothing.

**start\_body\_item** (*item*)

Called, by default, when keywords, messages or control structures start.

More specific `start_keyword()`, `start_message()`, `:meth:'start_for`, etc. can be implemented to visit only keywords, messages or specific control structures.

Can return explicit `False` to stop visiting. Default implementation does nothing.

**start\_break** (*break\_*)

Called when a BREAK element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_continue** (*continue\_*)

Called when a CONTINUE element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_for** (*for\_*)

Called when a FOR loop starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_for\_iteration** (*iteration*)

Called when a FOR loop iteration starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_if** (*if\_*)

Called when an IF/ELSE structure starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_if\_branch** (*branch*)

Called when an IF/ELSE branch starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_message** (*msg*)

Called when a message starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_return** (*return\_*)

Called when a RETURN element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_try** (*try\_*)

Called when a TRY/EXCEPT structure starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_try\_branch** (*branch*)

Called when TRY, EXCEPT, ELSE or FINALLY branches start.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_while** (*while\_*)

Called when a WHILE loop starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_while\_iteration** (*iteration*)

Called when a WHILE loop iteration starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**visit\_break** (*break\_*)

Visits BREAK elements.

**visit\_continue** (*continue\_*)

Visits CONTINUE elements.

**visit\_for** (*for\_*)

Implements traversing through FOR loops.

Can be overridden to allow modifying the passed in `for_` without calling `start_for()` or `end_for()` nor visiting body.

**visit\_for\_iteration** (*iteration*)

Implements traversing through single FOR loop iteration.

This is only used with the result side model because on the running side there are no iterations.

Can be overridden to allow modifying the passed in `iteration` without calling `start_for_iteration()` or `end_for_iteration()` nor visiting body.

**visit\_if** (*if\_*)

Implements traversing through IF/ELSE structures.

Notice that `if_` does not have any data directly. Actual IF/ELSE branches are in its body and visited using `visit_if_branch()`.

Can be overridden to allow modifying the passed in `if_` without calling `start_if()` or `end_if()` nor visiting branches.

**visit\_if\_branch** (*branch*)

Implements traversing through single IF/ELSE branch.

Can be overridden to allow modifying the passed in `branch` without calling `start_if_branch()` or `end_if_branch()` nor visiting body.

**visit\_keyword** (*kw*)

Implements traversing through keywords.

Can be overridden to allow modifying the passed in `kw` without calling `start_keyword()` or `end_keyword()` nor visiting the body of the keyword

**visit\_return** (*return\_*)

Visits a RETURN elements.

**visit\_suite** (*suite*)

Implements traversing through suites.

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

**visit\_test** (*test*)

Implements traversing through tests.

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

**visit\_try** (*try\_*)

Implements traversing through TRY/EXCEPT structures.

This method is used with the TRY/EXCEPT root element. Actual TRY, EXCEPT, ELSE and FINALLY branches are visited separately using `visit_try_branch()`.



**visit\_try\_branch** (*branch*)

Visits individual TRY, EXCEPT, ELSE and FINALLY branches.

**visit\_while** (*while\_*)

Implements traversing through WHILE loops.

Can be overridden to allow modifying the passed in *while\_* without calling `start_while()` or `end_while()` nor visiting body.

**visit\_while\_iteration** (*iteration*)

Implements traversing through single WHILE loop iteration.

This is only used with the result side model because on the running side there are no iterations.

Can be overridden to allow modifying the passed in *iteration* without calling `start_while_iteration()` or `end_while_iteration()` nor visiting body.

**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*, *rpa=False*)

Bases: `robot.model.visitor.SuiteVisitor`

**merge** (*merged*)

**start\_suite** (*suite*)

Called when a suite starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

**end\_suite** (*suite*)

Called when a suite ends. Default implementation does nothing.

**visit\_test** (*test*)

Implements traversing through tests.

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

**end\_body\_item** (*item*)

Called, by default, when keywords, messages or control structures end.

More specific `end_keyword()`, `end_message()`, `:meth:'end_for'`, etc. can be implemented to visit only keywords, messages or specific control structures.

Default implementation does nothing.

**end\_break** (*break\_*)

Called when a BREAK element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_continue** (*continue\_*)

Called when a CONTINUE element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_for** (*for\_*)

Called when a FOR loop ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_for\_iteration** (*iteration*)

Called when a FOR loop iteration ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_if** (*if\_*)

Called when an IF/ELSE structure ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_if\_branch** (*branch*)

Called when an IF/ELSE branch ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_keyword** (*keyword*)

Called when a keyword ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_message** (*msg*)

Called when a message ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_return** (*return\_*)

Called when a RETURN element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_test** (*test*)

Called when a test ends. Default implementation does nothing.

**end\_try** (*try\_*)

Called when a TRY/EXCEPT structure ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_try\_branch** (*branch*)

Called when TRY, EXCEPT, ELSE and FINALLY branches end.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_while** (*while\_*)

Called when a WHILE loop ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_while\_iteration** (*iteration*)

Called when a WHILE loop iteration ends.

By default, calls `end_body_item()` which, by default, does nothing.

**start\_body\_item** (*item*)

Called, by default, when keywords, messages or control structures start.

More specific `start_keyword()`, `start_message()`, `:meth:'start_for`, etc. can be implemented to visit only keywords, messages or specific control structures.

Can return explicit `False` to stop visiting. Default implementation does nothing.

**start\_break** (*break\_*)

Called when a BREAK element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_continue** (*continue\_*)

Called when a CONTINUE element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_for** (*for\_*)

Called when a FOR loop starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_for\_iteration** (*iteration*)

Called when a FOR loop iteration starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_if** (*if\_*)

Called when an IF/ELSE structure starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_if\_branch** (*branch*)

Called when an IF/ELSE branch starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_keyword** (*keyword*)

Called when a keyword starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_message** (*msg*)

Called when a message starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_return** (*return\_*)

Called when a RETURN element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_test** (*test*)

Called when a test starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

**start\_try** (*try\_*)

Called when a TRY/EXCEPT structure starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_try\_branch** (*branch*)

Called when TRY, EXCEPT, ELSE or FINALLY branches start.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_while** (*while\_*)

Called when a WHILE loop starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_while\_iteration** (*iteration*)

Called when a WHILE loop iteration starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**visit\_break** (*break\_*)

Visits BREAK elements.

**visit\_continue** (*continue\_*)

Visits CONTINUE elements.

**visit\_for** (*for\_*)

Implements traversing through FOR loops.

Can be overridden to allow modifying the passed in `for_` without calling `start_for()` or `end_for()` nor visiting body.

**visit\_for\_iteration** (*iteration*)

Implements traversing through single FOR loop iteration.

This is only used with the result side model because on the running side there are no iterations.

Can be overridden to allow modifying the passed in `iteration` without calling `start_for_iteration()` or `end_for_iteration()` nor visiting body.

**visit\_if** (*if\_*)

Implements traversing through IF/ELSE structures.

Notice that `if_` does not have any data directly. Actual IF/ELSE branches are in its `body` and visited using `visit_if_branch()`.

Can be overridden to allow modifying the passed in `if_` without calling `start_if()` or `end_if()` nor visiting branches.

**visit\_if\_branch** (*branch*)

Implements traversing through single IF/ELSE branch.

Can be overridden to allow modifying the passed in `branch` without calling `start_if_branch()` or `end_if_branch()` nor visiting body.

**visit\_keyword** (*kw*)

Implements traversing through keywords.

Can be overridden to allow modifying the passed in `kw` without calling `start_keyword()` or `end_keyword()` nor visiting the body of the keyword

**visit\_message**(*msg*)

Implements visiting messages.

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

**visit\_return**(*return\_*)

Visits a RETURN elements.

**visit\_suite**(*suite*)

Implements traversing through suites.

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

**visit\_try**(*try\_*)

Implements traversing through TRY/EXCEPT structures.

This method is used with the TRY/EXCEPT root element. Actual TRY, EXCEPT, ELSE and FINALLY branches are visited separately using `visit_try_branch()`.

**visit\_try\_branch**(*branch*)

Visits individual TRY, EXCEPT, ELSE and FINALLY branches.

**visit\_while**(*while\_*)

Implements traversing through WHILE loops.

Can be overridden to allow modifying the passed in `while_` without calling `start_while()` or `end_while()` nor visiting body.

**visit\_while\_iteration**(*iteration*)

Implements traversing through single WHILE loop iteration.

This is only used with the result side model because on the running side there are no iterations.

Can be overridden to allow modifying the passed in `iteration` without calling `start_while_iteration()` or `end_while_iteration()` nor visiting body.

## robot.result.messagefilter module

**class** `robot.result.messagefilter.MessageFilter`(*log\_level=None*)

Bases: `robot.model.visitor.SuiteVisitor`

**start\_suite**(*suite*)

Called when a suite starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

**start\_keyword**(*keyword*)

Called when a keyword starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**end\_body\_item**(*item*)

Called, by default, when keywords, messages or control structures end.

More specific `end_keyword()`, `end_message()`, `:meth:'end_for'`, etc. can be implemented to visit only keywords, messages or specific control structures.

Default implementation does nothing.

**end\_break** (*break\_*)

Called when a BREAK element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_continue** (*continue\_*)

Called when a CONTINUE element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_for** (*for\_*)

Called when a FOR loop ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_for\_iteration** (*iteration*)

Called when a FOR loop iteration ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_if** (*if\_*)

Called when an IF/ELSE structure ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_if\_branch** (*branch*)

Called when an IF/ELSE branch ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_keyword** (*keyword*)

Called when a keyword ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_message** (*msg*)

Called when a message ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_return** (*return\_*)

Called when a RETURN element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_suite** (*suite*)

Called when a suite ends. Default implementation does nothing.

**end\_test** (*test*)

Called when a test ends. Default implementation does nothing.

**end\_try** (*try\_*)

Called when a TRY/EXCEPT structure ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_try\_branch** (*branch*)

Called when TRY, EXCEPT, ELSE and FINALLY branches end.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_while** (*while\_*)

Called when a WHILE loop ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_while\_iteration** (*iteration*)

Called when a WHILE loop iteration ends.

By default, calls `end_body_item()` which, by default, does nothing.

**start\_body\_item** (*item*)

Called, by default, when keywords, messages or control structures start.

More specific `start_keyword()`, `start_message()`, `:meth:'start_for`, etc. can be implemented to visit only keywords, messages or specific control structures.

Can return explicit `False` to stop visiting. Default implementation does nothing.

**start\_break** (*break\_*)

Called when a BREAK element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_continue** (*continue\_*)

Called when a CONTINUE element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_for** (*for\_*)

Called when a FOR loop starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_for\_iteration** (*iteration*)

Called when a FOR loop iteration starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_if** (*if\_*)

Called when an IF/ELSE structure starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_if\_branch** (*branch*)

Called when an IF/ELSE branch starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_message** (*msg*)

Called when a message starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_return** (*return\_*)

Called when a RETURN element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_test** (*test*)

Called when a test starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

**start\_try** (*try\_*)

Called when a TRY/EXCEPT structure starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_try\_branch** (*branch*)

Called when TRY, EXCEPT, ELSE or FINALLY branches start.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_while** (*while\_*)

Called when a WHILE loop starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_while\_iteration** (*iteration*)

Called when a WHILE loop iteration starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**visit\_break** (*break\_*)

Visits BREAK elements.

**visit\_continue** (*continue\_*)

Visits CONTINUE elements.

**visit\_for** (*for\_*)

Implements traversing through FOR loops.

Can be overridden to allow modifying the passed in `for_` without calling `start_for()` or `end_for()` nor visiting body.

**visit\_for\_iteration** (*iteration*)

Implements traversing through single FOR loop iteration.

This is only used with the result side model because on the running side there are no iterations.

Can be overridden to allow modifying the passed in `iteration` without calling `start_for_iteration()` or `end_for_iteration()` nor visiting body.

**visit\_if** (*if\_*)

Implements traversing through IF/ELSE structures.

Notice that `if_` does not have any data directly. Actual IF/ELSE branches are in its `body` and visited using `visit_if_branch()`.

Can be overridden to allow modifying the passed in `if_` without calling `start_if()` or `end_if()` nor visiting branches.

**visit\_if\_branch** (*branch*)

Implements traversing through single IF/ELSE branch.



Can be overridden to allow modifying the passed in `branch` without calling `start_if_branch()` or `end_if_branch()` nor visiting body.

**visit\_keyword** (*kw*)

Implements traversing through keywords.

Can be overridden to allow modifying the passed in `kw` without calling `start_keyword()` or `end_keyword()` nor visiting the body of the keyword

**visit\_message** (*msg*)

Implements visiting messages.

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

**visit\_return** (*return\_*)

Visits a RETURN elements.

**visit\_suite** (*suite*)

Implements traversing through suites.

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

**visit\_test** (*test*)

Implements traversing through tests.

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

**visit\_try** (*try\_*)

Implements traversing through TRY/EXCEPT structures.

This method is used with the TRY/EXCEPT root element. Actual TRY, EXCEPT, ELSE and FINALLY branches are visited separately using `visit_try_branch()`.

**visit\_try\_branch** (*branch*)

Visits individual TRY, EXCEPT, ELSE and FINALLY branches.

**visit\_while** (*while\_*)

Implements traversing through WHILE loops.

Can be overridden to allow modifying the passed in `while_` without calling `start_while()` or `end_while()` nor visiting body.

**visit\_while\_iteration** (*iteration*)

Implements traversing through single WHILE loop iteration.

This is only used with the result side model because on the running side there are no iterations.

Can be overridden to allow modifying the passed in `iteration` without calling `start_while_iteration()` or `end_while_iteration()` nor visiting body.

## robot.result.model module

Module implementing result related model objects.

During test execution these objects are created internally by various runners. At that time they can inspected and modified by [listeners](#).

When results are parsed from XML output files after execution to be able to create logs and reports, these objects are created by the `ExecutionResult()` factory method. At that point they can be inspected and modified by `pre-Rebot` modifiers.

The `ExecutionResult()` factory method can also be used by custom scripts and tools. In such usage it is often easiest to inspect and modify these objects using the `visitor interface`.

```
class robot.result.model.Body (parent=None, items=None)
    Bases: robot.model.body.Body

    append (item)
        S.append(value) – append value to the end of the sequence

    break_class
        alias of Break

    clear () → None – remove all items from S

    continue_class
        alias of Continue

    count (value) → integer – return number of occurrences of value

    create

    create_break (*args, **kwargs)

    create_continue (*args, **kwargs)

    create_for (*args, **kwargs)

    create_if (*args, **kwargs)

    create_keyword (*args, **kwargs)

    create_message (*args, **kwargs)

    create_return (*args, **kwargs)

    create_try (*args, **kwargs)

    create_while (*args, **kwargs)

    extend (items)
        S.extend(iterable) – extend sequence by appending elements from the iterable

    filter (keywords=None, messages=None, predicate=None)
        Filter body items based on type and/or custom predicate.
```

To include or exclude items based on types, give matching arguments `True` or `False` values. For example, to include only keywords, use `body.filter(keywords=True)` and to exclude messages use `body.filter(messages=False)`. Including and excluding by types at the same time is not supported and filtering my messages is supported only if the `Body` object actually supports messages.

Custom `predicate` is a callable getting each body item as an argument that must return `True/False` depending on should the item be included or not.

Selected items are returned as a list and the original body is not modified.

It was earlier possible to filter also based on `FOR` and `IF` types. That support was removed in RF 5.0 because it was not considered useful in general and because adding support for all new control structures would have required extra work. To exclude all control structures, use `body.filter(keywords=True, messages=True)` and to only include them use `body.filter(keywords=False, messages=False)`. For more detailed filtering it is possible to use `predicate`.

**flatten()**  
Return steps so that IF and TRY structures are flattened.  
Basically the IF/ELSE and TRY/EXCEPT root elements are replaced with their branches. This is how they are shown in log files.

**for\_class**  
alias of *For*

**if\_class**  
alias of *If*

**index** (*value* [, *start* [, *stop* ]]) → integer – return first index of value.  
Raises ValueError if the value is not present.  
Supporting start and stop arguments is optional, but recommended.

**insert** (*index*, *item*)  
S.insert(*index*, *value*) – insert value before index

**keyword\_class**  
alias of *Keyword*

**message\_class**  
alias of *Message*

**pop** ([*index*]) → item – remove and return item at index (default last).  
Raise IndexError if list is empty or index is out of range.

**classmethod register** (*item\_class*)  
Register a virtual subclass of an ABC.  
Returns the subclass, to allow usage as a class decorator.

**remove** (*value*)  
S.remove(*value*) – remove first occurrence of value. Raise ValueError if the value is not present.

**return\_class**  
alias of *Return*

**reverse** ()  
S.reverse() – reverse *IN PLACE*

**sort** ()

**try\_class**  
alias of *Try*

**visit** (*visitor*)

**while\_class**  
alias of *While*

**class** robot.result.model.**Branches** (*branch\_class*, *parent=None*, *items=None*)  
Bases: *robot.model.body.Banches*

**append** (*item*)  
S.append(*value*) – append value to the end of the sequence

**branch\_class**

**break\_class** = None

**clear** () → None – remove all items from S

**continue\_class** = None

**count** (*value*) → integer – return number of occurrences of value

**create**

**create\_branch** (*\*args*, *\*\*kwargs*)

**create\_break** (*\*args*, *\*\*kwargs*)

**create\_continue** (*\*args*, *\*\*kwargs*)

**create\_for** (*\*args*, *\*\*kwargs*)

**create\_if** (*\*args*, *\*\*kwargs*)

**create\_keyword** (*\*args*, *\*\*kwargs*)

**create\_message** (*\*args*, *\*\*kwargs*)

**create\_return** (*\*args*, *\*\*kwargs*)

**create\_try** (*\*args*, *\*\*kwargs*)

**create\_while** (*\*args*, *\*\*kwargs*)

**extend** (*items*)

S.extend(iterable) – extend sequence by appending elements from the iterable

**filter** (*keywords=None*, *messages=None*, *predicate=None*)

Filter body items based on type and/or custom predicate.

To include or exclude items based on types, give matching arguments True or False values. For example, to include only keywords, use `body.filter(keywords=True)` and to exclude messages use `body.filter(messages=False)`. Including and excluding by types at the same time is not supported and filtering my messages is supported only if the Body object actually supports messages.

Custom *predicate* is a callable getting each body item as an argument that must return True/False depending on should the item be included or not.

Selected items are returned as a list and the original body is not modified.

It was earlier possible to filter also based on FOR and IF types. That support was removed in RF 5.0 because it was not considered useful in general and because adding support for all new control structures would have required extra work. To exclude all control structures, use `body.filter(keywords=True, messages=True)` and to only include them use `body.filter(keywords=False, messages=False)`. For more detailed filtering it is possible to use *predicate*.

**flatten** ()

Return steps so that IF and TRY structures are flattened.

Basically the IF/ELSE and TRY/EXCEPT root elements are replaced with their branches. This is how they are shown in log files.

**for\_class** = None

**if\_class** = None

**index** (*value*[, *start*[, *stop*]]) → integer – return first index of value.

Raises ValueError if the value is not present.

Supporting start and stop arguments is optional, but recommended.

**insert** (*index*, *item*)

S.insert(index, value) – insert value before index

```

keyword_class
    alias of Keyword

message_class
    alias of Message

pop ([index]) → item – remove and return item at index (default last).
    Raise IndexError if list is empty or index is out of range.

classmethod register (item_class)
    Register a virtual subclass of an ABC.

    Returns the subclass, to allow usage as a class decorator.

remove (value)
    S.remove(value) – remove first occurrence of value. Raise ValueError if the value is not present.

return_class = None

reverse ()
    S.reverse() – reverse IN PLACE

sort ()

try_class = None

visit (visitor)

while_class = None

class robot.result.model.Iterations (iteration_class, parent=None, items=None)
    Bases: robot.model.body.BaseBody

    iteration_class

    create_iteration (*args, **kwargs)

    append (item)
        S.append(value) – append value to the end of the sequence

    break_class = None

    clear () → None – remove all items from S

    continue_class = None

    count (value) → integer – return number of occurrences of value

    create

    create_break (*args, **kwargs)

    create_continue (*args, **kwargs)

    create_for (*args, **kwargs)

    create_if (*args, **kwargs)

    create_keyword (*args, **kwargs)

    create_message (*args, **kwargs)

    create_return (*args, **kwargs)

    create_try (*args, **kwargs)

    create_while (*args, **kwargs)

```

**extend** (*items*)

S.extend(iterable) – extend sequence by appending elements from the iterable

**filter** (*keywords=None, messages=None, predicate=None*)

Filter body items based on type and/or custom predicate.

To include or exclude items based on types, give matching arguments `True` or `False` values. For example, to include only keywords, use `body.filter(keywords=True)` and to exclude messages use `body.filter(messages=False)`. Including and excluding by types at the same time is not supported and filtering my messages is supported only if the `Body` object actually supports messages.

Custom `predicate` is a callable getting each body item as an argument that must return `True/False` depending on should the item be included or not.

Selected items are returned as a list and the original body is not modified.

It was earlier possible to filter also based on `FOR` and `IF` types. That support was removed in RF 5.0 because it was not considered useful in general and because adding support for all new control structures would have required extra work. To exclude all control structures, use `body.filter(keywords=True, messages=True)` and to only include them use `body.filter(keywords=False, messages=False)`. For more detailed filtering it is possible to use `predicate`.

**flatten** ()

Return steps so that `IF` and `TRY` structures are flattened.

Basically the `IF/ELSE` and `TRY/EXCEPT` root elements are replaced with their branches. This is how they are shown in log files.

**for\_class** = `None`

**if\_class** = `None`

**index** (*value* [, *start* [, *stop* ]]) → integer – return first index of value.

Raises `ValueError` if the value is not present.

Supporting `start` and `stop` arguments is optional, but recommended.

**insert** (*index, item*)

S.insert(index, value) – insert value before index

**keyword\_class**

alias of *Keyword*

**message\_class**

alias of *Message*

**pop** ([*index*]) → item – remove and return item at index (default last).

Raise `IndexError` if list is empty or index is out of range.

**classmethod register** (*item\_class*)

Register a virtual subclass of an ABC.

Returns the subclass, to allow usage as a class decorator.

**remove** (*value*)

S.remove(value) – remove first occurrence of value. Raise `ValueError` if the value is not present.

**return\_class** = `None`

**reverse** ()

S.reverse() – reverse *IN PLACE*

**sort** ()

```

    try_class = None
    visit(visitor)
    while_class = None
class robot.result.model.Message(message="", level='INFO', html=False, timestamp=None,
                                parent=None)
    Bases: robot.model.message.Message
    BREAK = 'BREAK'
    CONTINUE = 'CONTINUE'
    ELSE = 'ELSE'
    ELSE_IF = 'ELSE IF'
    EXCEPT = 'EXCEPT'
    FINALLY = 'FINALLY'
    FOR = 'FOR'
    IF = 'IF'
    IF_ELSE_ROOT = 'IF/ELSE ROOT'
    ITERATION = 'ITERATION'
    KEYWORD = 'KEYWORD'
    MESSAGE = 'MESSAGE'
    RETURN = 'RETURN'
    SETUP = 'SETUP'
    TEARDOWN = 'TEARDOWN'
    TRY = 'TRY'
    TRY_EXCEPT_ROOT = 'TRY/EXCEPT ROOT'
    WHILE = 'WHILE'
    config(**attributes)
        Configure model object with given attributes.

        obj.config(name='Example', doc='Something') is equivalent to setting obj.name =
        'Example' and obj.doc = 'Something'.

        New in Robot Framework 4.0.
    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.
    deepcopy(**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.

**has\_setup**

**has\_teardown**

**html**

**html\_message**

Returns the message content as HTML.

**id**

Item id in format like s1-t3-k1.

See `TestSuite.id` for more information.

**level**

**message**

**parent**

**repr\_args** = ('message', 'level')

**timestamp**

**type** = 'MESSAGE'

**visit** (*visitor*)

*Visitor interface* entry-point.

**class** robot.result.model.StatusMixin

Bases: object

**PASS** = 'PASS'

**FAIL** = 'FAIL'

**SKIP** = 'SKIP'

**NOT\_RUN** = 'NOT RUN'

**NOT\_SET** = 'NOT SET'

**elapsedtime**

Total execution time in milliseconds.

**passed**

True when status is 'PASS', False otherwise.

**failed**

True when status is 'FAIL', False otherwise.

**skipped**

True when status is 'SKIP', False otherwise.

Setting to False value is ambiguous and raises an exception.

**not\_run**

True when status is 'NOT RUN', False otherwise.

Setting to False value is ambiguous and raises an exception.

**class** robot.result.model.ForIteration (*variables=None, status='FAIL', starttime=None, end-time=None, doc="", parent=None*)

Bases: `robot.model.body.BodyItem`, `robot.result.model.StatusMixin`, `robot.result.model.deprecation.DeprecatedAttributesMixin`



Represents one FOR loop iteration.

**type** = 'ITERATION'

**body\_class**  
alias of *Body*

**repr\_args** = ('variables',)

**variables**

**parent**

**status**

**starttime**

**endtime**

**doc**

**body**

**visit** (*visitor*)

**name**  
Deprecated.

**BREAK** = 'BREAK'

**CONTINUE** = 'CONTINUE'

**ELSE** = 'ELSE'

**ELSE\_IF** = 'ELSE IF'

**EXCEPT** = 'EXCEPT'

**FAIL** = 'FAIL'

**FINALLY** = 'FINALLY'

**FOR** = 'FOR'

**IF** = 'IF'

**IF\_ELSE\_ROOT** = 'IF/ELSE ROOT'

**ITERATION** = 'ITERATION'

**KEYWORD** = 'KEYWORD'

**MESSAGE** = 'MESSAGE'

**NOT\_RUN** = 'NOT RUN'

**NOT\_SET** = 'NOT SET'

**PASS** = 'PASS'

**RETURN** = 'RETURN'

**SETUP** = 'SETUP'

**SKIP** = 'SKIP'

**TEARDOWN** = 'TEARDOWN'

**TRY** = 'TRY'

**TRY\_EXCEPT\_ROOT** = 'TRY/EXCEPT ROOT'

**WHILE** = 'WHILE'

**args**  
Deprecated.

**assign**  
Deprecated.

**config** (\*\*attributes)  
Configure model object with given attributes.  
  
obj.config(name='Example', doc='Something') is equivalent to setting obj.name = 'Example' and obj.doc = 'Something'.  
  
New in Robot Framework 4.0.

**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.

**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.

**elapsedtime**  
Total execution time in milliseconds.

**failed**  
True when [status](#) is 'FAIL', False otherwise.

**has\_setup**

**has\_teardown**

**id**  
Item id in format like s1-t3-k1.  
  
See [TestSuite.id](#) for more information.

**kwname**  
Deprecated.

**libname**  
Deprecated.

**message**  
Deprecated.

**not\_run**  
True when [status](#) is 'NOT RUN', False otherwise.  
  
Setting to False value is ambiguous and raises an exception.

**passed**  
True when [status](#) is 'PASS', False otherwise.

**skipped**

True when *status* is 'SKIP', False otherwise.

Setting to False value is ambiguous and raises an exception.

**tags**

Deprecated.

**timeout**

Deprecated.

```
class robot.result.model.For(variables=(), flavor='IN', values=(), status='FAIL', start-
                             time=None, endtime=None, doc="", parent=None)
Bases: robot.model.control.For, robot.result.model.StatusMixin, robot.result.
modeldeprecation.DeprecatedAttributesMixin
```

**iterations\_class**

alias of *Iterations*

**iteration\_class**

alias of *ForIteration*

**status****starttime****endtime****doc****body****name**

Deprecated.

**BREAK** = 'BREAK'

**CONTINUE** = 'CONTINUE'

**ELSE** = 'ELSE'

**ELSE\_IF** = 'ELSE IF'

**EXCEPT** = 'EXCEPT'

**FAIL** = 'FAIL'

**FINALLY** = 'FINALLY'

**FOR** = 'FOR'

**IF** = 'IF'

**IF\_ELSE\_ROOT** = 'IF/ELSE ROOT'

**ITERATION** = 'ITERATION'

**KEYWORD** = 'KEYWORD'

**MESSAGE** = 'MESSAGE'

**NOT\_RUN** = 'NOT RUN'

**NOT\_SET** = 'NOT SET'

**PASS** = 'PASS'

**RETURN** = 'RETURN'

**SETUP** = 'SETUP'

**SKIP** = 'SKIP'

**TEARDOWN** = 'TEARDOWN'

**TRY** = 'TRY'

**TRY\_EXCEPT\_ROOT** = 'TRY/EXCEPT ROOT'

**WHILE** = 'WHILE'

**args**  
Deprecated.

**assign**  
Deprecated.

**body\_class**  
alias of `robot.model.body.Body`

**config** (*\*\*attributes*)  
Configure model object with given attributes.  
  
obj.config(name='Example', doc='Something') is equivalent to setting obj.name = 'Example' and obj.doc = 'Something'.  
  
New in Robot Framework 4.0.

**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.

**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.

**elapsedtime**  
Total execution time in milliseconds.

**failed**  
True when `status` is 'FAIL', False otherwise.

**flavor**

**has\_setup**

**has\_teardown**

**id**  
Item id in format like s1-t3-k1.

See `TestSuite.id` for more information.

**keywords**  
Deprecated since Robot Framework 4.0. Use `body` instead.

```

kwname
    Deprecated.

libname
    Deprecated.

message
    Deprecated.

not_run
    True when status is 'NOT RUN', False otherwise.

    Setting to False value is ambiguous and raises an exception.

parent

passed
    True when status is 'PASS', False otherwise.

repr_args = ('variables', 'flavor', 'values')

skipped
    True when status is 'SKIP', False otherwise.

    Setting to False value is ambiguous and raises an exception.

tags
    Deprecated.

timeout
    Deprecated.

type = 'FOR'

values

variables

visit (visitor)

class robot.result.model.WhileIteration(status='FAIL', starttime=None, endtime=None,
                                         doc="", parent=None)
    Bases: robot.model.body.BodyItem, robot.result.model.StatusMixin, robot.
           result.model.deprecation.DeprecatedAttributesMixin
    Represents one WHILE loop iteration.

    type = 'ITERATION'

body_class
    alias of Body

parent

status

starttime

endtime

doc

body

visit (visitor)

name
    Deprecated.

```

```
BREAK = 'BREAK'
CONTINUE = 'CONTINUE'
ELSE = 'ELSE'
ELSE_IF = 'ELSE IF'
EXCEPT = 'EXCEPT'
FAIL = 'FAIL'
FINALLY = 'FINALLY'
FOR = 'FOR'
IF = 'IF'
IF_ELSE_ROOT = 'IF/ELSE ROOT'
ITERATION = 'ITERATION'
KEYWORD = 'KEYWORD'
MESSAGE = 'MESSAGE'
NOT_RUN = 'NOT RUN'
NOT_SET = 'NOT SET'
PASS = 'PASS'
RETURN = 'RETURN'
SETUP = 'SETUP'
SKIP = 'SKIP'
TEARDOWN = 'TEARDOWN'
TRY = 'TRY'
TRY_EXCEPT_ROOT = 'TRY/EXCEPT ROOT'
WHILE = 'WHILE'
```

**args**  
Deprecated.

**assign**  
Deprecated.

**config** (*\*\*attributes*)  
Configure model object with given attributes.

`obj.config(name='Example', doc='Something')` is equivalent to setting `obj.name = 'Example'` and `obj.doc = 'Something'`.

New in Robot Framework 4.0.

**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.

**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.

**elapsedtime**

Total execution time in milliseconds.

**failed**

True when *status* is 'FAIL', False otherwise.

**has\_setup**

**has\_teardown**

**id**

Item id in format like s1-t3-k1.

See `TestSuite.id` for more information.

**kwname**

Deprecated.

**libname**

Deprecated.

**message**

Deprecated.

**not\_run**

True when *status* is 'NOT RUN', False otherwise.

Setting to False value is ambiguous and raises an exception.

**passed**

True when *status* is 'PASS', False otherwise.

**repr\_args** = ()

**skipped**

True when *status* is 'SKIP', False otherwise.

Setting to False value is ambiguous and raises an exception.

**tags**

Deprecated.

**timeout**

Deprecated.

**class** `robot.result.model.While` (*condition=None, limit=None, parent=None, status='FAIL', starttime=None, endtime=None, doc=""*)

Bases: `robot.model.control.While`, `robot.result.model.StatusMixin`, `robot.result.model.deprecation.DeprecatedAttributesMixin`

**iterations\_class**

alias of `Iterations`

**iteration\_class**

alias of `WhileIteration`

**status**

**starttime**  
**endtime**  
**doc**  
**body**  
**name**  
    Deprecated.  
**BREAK** = 'BREAK'  
**CONTINUE** = 'CONTINUE'  
**ELSE** = 'ELSE'  
**ELSE\_IF** = 'ELSE IF'  
**EXCEPT** = 'EXCEPT'  
**FAIL** = 'FAIL'  
**FINALLY** = 'FINALLY'  
**FOR** = 'FOR'  
**IF** = 'IF'  
**IF\_ELSE\_ROOT** = 'IF/ELSE ROOT'  
**ITERATION** = 'ITERATION'  
**KEYWORD** = 'KEYWORD'  
**MESSAGE** = 'MESSAGE'  
**NOT\_RUN** = 'NOT RUN'  
**NOT\_SET** = 'NOT SET'  
**PASS** = 'PASS'  
**RETURN** = 'RETURN'  
**SETUP** = 'SETUP'  
**SKIP** = 'SKIP'  
**TEARDOWN** = 'TEARDOWN'  
**TRY** = 'TRY'  
**TRY\_EXCEPT\_ROOT** = 'TRY/EXCEPT ROOT'  
**WHILE** = 'WHILE'  
**args**  
    Deprecated.  
**assign**  
    Deprecated.  
**body\_class**  
    alias of *robot.model.body.Body*  
**condition**



**config** (\*\*attributes)

Configure model object with given attributes.

`obj.config(name='Example', doc='Something')` is equivalent to setting `obj.name = 'Example'` and `obj.doc = 'Something'`.

New in Robot Framework 4.0.

**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.

**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.

**elapsedtime**

Total execution time in milliseconds.

**failed**

True when [`status`](#) is 'FAIL', False otherwise.

**has\_setup****has\_teardown****id**

Item id in format like `s1-t3-k1`.

See [`TestSuite.id`](#) for more information.

**kwname**

Deprecated.

**libname**

Deprecated.

**limit****message**

Deprecated.

**not\_run**

True when [`status`](#) is 'NOT RUN', False otherwise.

Setting to False value is ambiguous and raises an exception.

**parent****passed**

True when [`status`](#) is 'PASS', False otherwise.

**repr\_args** = ('condition', 'limit')

**skipped**

True when *status* is 'SKIP', False otherwise.

Setting to False value is ambiguous and raises an exception.

**tags**

Deprecated.

**timeout**

Deprecated.

**type** = 'WHILE'

**visit** (*visitor*)

**class** robot.result.model.IfBranch (*type='IF', condition=None, status='FAIL', starttime=None, endtime=None, doc="", parent=None*)

Bases: *robot.model.control.IfBranch, robot.result.model.StatusMixin, robot.result.model.deprecation.DeprecatedAttributesMixin*

**body\_class**

alias of *Body*

**status****starttime****endtime****doc****name**

Deprecated.

**BREAK** = 'BREAK'

**CONTINUE** = 'CONTINUE'

**ELSE** = 'ELSE'

**ELSE\_IF** = 'ELSE IF'

**EXCEPT** = 'EXCEPT'

**FAIL** = 'FAIL'

**FINALLY** = 'FINALLY'

**FOR** = 'FOR'

**IF** = 'IF'

**IF\_ELSE\_ROOT** = 'IF/ELSE ROOT'

**ITERATION** = 'ITERATION'

**KEYWORD** = 'KEYWORD'

**MESSAGE** = 'MESSAGE'

**NOT\_RUN** = 'NOT RUN'

**NOT\_SET** = 'NOT SET'

**PASS** = 'PASS'

**RETURN** = 'RETURN'

**SETUP** = 'SETUP'

```

SKIP = 'SKIP'
TEARDOWN = 'TEARDOWN'
TRY = 'TRY'
TRY_EXCEPT_ROOT = 'TRY/EXCEPT ROOT'
WHILE = 'WHILE'

args
    Deprecated.

assign
    Deprecated.

body

condition

config (**attributes)
    Configure model object with given attributes.

    obj.config(name='Example', doc='Something') is equivalent to setting obj.name =
    'Example' and obj.doc = 'Something'.

    New in Robot Framework 4.0.

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.

deepcopy (**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.

elapsedtime
    Total execution time in milliseconds.

failed
    True when status is 'FAIL', False otherwise.

has_setup

has_teardown

id
    Branch id omits IF/ELSE root from the parent id part.

kname
    Deprecated.

libname
    Deprecated.

```

**message**

Deprecated.

**not\_run**

True when *status* is 'NOT RUN', False otherwise.

Setting to False value is ambiguous and raises an exception.

**parent**

**passed**

True when *status* is 'PASS', False otherwise.

**repr\_args** = ('type', 'condition')

**skipped**

True when *status* is 'SKIP', False otherwise.

Setting to False value is ambiguous and raises an exception.

**tags**

Deprecated.

**timeout**

Deprecated.

**type**

**visit** (*visitor*)

**class** robot.result.model.If(*status='FAIL', starttime=None, endtime=None, doc="", parent=None*)

Bases: *robot.model.control.If*, *robot.result.model.StatusMixin*, *robot.result.model.deprecation.DeprecatedAttributesMixin*

**branch\_class**

alias of *IfBranch*

**branches\_class**

alias of *Branches*

**status**

**starttime**

**endtime**

**doc**

**BREAK** = 'BREAK'

**CONTINUE** = 'CONTINUE'

**ELSE** = 'ELSE'

**ELSE\_IF** = 'ELSE IF'

**EXCEPT** = 'EXCEPT'

**FAIL** = 'FAIL'

**FINALLY** = 'FINALLY'

**FOR** = 'FOR'

**IF** = 'IF'

**IF\_ELSE\_ROOT** = 'IF/ELSE ROOT'

```

ITERATION = 'ITERATION'
KEYWORD = 'KEYWORD'
MESSAGE = 'MESSAGE'
NOT_RUN = 'NOT RUN'
NOT_SET = 'NOT SET'
PASS = 'PASS'
RETURN = 'RETURN'
SETUP = 'SETUP'
SKIP = 'SKIP'
TEARDOWN = 'TEARDOWN'
TRY = 'TRY'
TRY_EXCEPT_ROOT = 'TRY/EXCEPT ROOT'
WHILE = 'WHILE'

args
    Deprecated.

assign
    Deprecated.

body

config (**attributes)
    Configure model object with given attributes.

    obj.config(name='Example', doc='Something') is equivalent to setting obj.name =
    'Example' and obj.doc = 'Something'.

    New in Robot Framework 4.0.

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.

deepcopy (**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.

elapsedtime
    Total execution time in milliseconds.

failed
    True when status is 'FAIL', False otherwise.

has_setup

```

**has\_teardown**

**id**  
Root IF/ELSE id is always None.

**kwname**  
Deprecated.

**libname**  
Deprecated.

**message**  
Deprecated.

**name**  
Deprecated.

**not\_run**  
True when *status* is 'NOT RUN', False otherwise.  
  
Setting to False value is ambiguous and raises an exception.

**parent**

**passed**  
True when *status* is 'PASS', False otherwise.

**repr\_args** = ()

**skipped**  
True when *status* is 'SKIP', False otherwise.  
  
Setting to False value is ambiguous and raises an exception.

**tags**  
Deprecated.

**timeout**  
Deprecated.

**type** = 'IF/ELSE ROOT'

**visit** (*visitor*)

**class** robot.result.model.TryBranch(*type*='TRY', *patterns*=(), *pattern\_type*=None, *variable*=None, *status*='FAIL', *starttime*=None, *endtime*=None, *doc*="", *parent*=None)

Bases: *robot.model.control.TryBranch*, *robot.result.model.StatusMixin*, *robot.result.model.deprecation.DeprecatedAttributesMixin*

**body\_class**  
alias of *Body*

**status**

**starttime**

**endtime**

**doc**

**name**  
Deprecated.

**BREAK** = 'BREAK'

**CONTINUE** = 'CONTINUE'

```

ELSE = 'ELSE'
ELSE_IF = 'ELSE IF'
EXCEPT = 'EXCEPT'
FAIL = 'FAIL'
FINALLY = 'FINALLY'
FOR = 'FOR'
IF = 'IF'
IF_ELSE_ROOT = 'IF/ELSE ROOT'
ITERATION = 'ITERATION'
KEYWORD = 'KEYWORD'
MESSAGE = 'MESSAGE'
NOT_RUN = 'NOT RUN'
NOT_SET = 'NOT SET'
PASS = 'PASS'
RETURN = 'RETURN'
SETUP = 'SETUP'
SKIP = 'SKIP'
TEARDOWN = 'TEARDOWN'
TRY = 'TRY'
TRY_EXCEPT_ROOT = 'TRY/EXCEPT ROOT'
WHILE = 'WHILE'

```

**args**  
Deprecated.

**assign**  
Deprecated.

**body**

**config** (\*\*attributes)  
Configure model object with given attributes.

`obj.config(name='Example', doc='Something')` is equivalent to setting `obj.name = 'Example'` and `obj.doc = 'Something'`.

New in Robot Framework 4.0.

**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.

**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.

**elapsedtime**

Total execution time in milliseconds.

**failed**

True when `status` is 'FAIL', False otherwise.

**has\_setup**

**has\_teardown**

**id**

Branch id omits TRY/EXCEPT root from the parent id part.

**kwname**

Deprecated.

**libname**

Deprecated.

**message**

Deprecated.

**not\_run**

True when `status` is 'NOT RUN', False otherwise.

Setting to False value is ambiguous and raises an exception.

**parent**

**passed**

True when `status` is 'PASS', False otherwise.

**pattern\_type**

**patterns**

**repr\_args** = ('type', 'patterns', 'pattern\_type', 'variable')

**skipped**

True when `status` is 'SKIP', False otherwise.

Setting to False value is ambiguous and raises an exception.

**tags**

Deprecated.

**timeout**

Deprecated.

**type**

**variable**

**visit** (*visitor*)

**class** `robot.result.model.Try` (*status='FAIL', starttime=None, endtime=None, doc="", parent=None*)

Bases: `robot.model.control.Try`, `robot.result.model.StatusMixin`, `robot.result.model.deprecation.DeprecatedAttributesMixin`



**branch\_class**  
alias of *TryBranch*

**branches\_class**  
alias of *Branches*

**status**

**starttime**

**endtime**

**doc**

**BREAK** = 'BREAK'

**CONTINUE** = 'CONTINUE'

**ELSE** = 'ELSE'

**ELSE\_IF** = 'ELSE IF'

**EXCEPT** = 'EXCEPT'

**FAIL** = 'FAIL'

**FINALLY** = 'FINALLY'

**FOR** = 'FOR'

**IF** = 'IF'

**IF\_ELSE\_ROOT** = 'IF/ELSE ROOT'

**ITERATION** = 'ITERATION'

**KEYWORD** = 'KEYWORD'

**MESSAGE** = 'MESSAGE'

**NOT\_RUN** = 'NOT RUN'

**NOT\_SET** = 'NOT SET'

**PASS** = 'PASS'

**RETURN** = 'RETURN'

**SETUP** = 'SETUP'

**SKIP** = 'SKIP'

**TEARDOWN** = 'TEARDOWN'

**TRY** = 'TRY'

**TRY\_EXCEPT\_ROOT** = 'TRY/EXCEPT ROOT'

**WHILE** = 'WHILE'

**args**  
Deprecated.

**assign**  
Deprecated.

**body**

**config** (*\*\*attributes*)

Configure model object with given attributes.

`obj.config(name='Example', doc='Something')` is equivalent to setting `obj.name = 'Example'` and `obj.doc = 'Something'`.

New in Robot Framework 4.0.

**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.

**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.

**elapsedtime**

Total execution time in milliseconds.

**else\_branch****except\_branches****failed**

True when *status* is 'FAIL', False otherwise.

**finally\_branch****has\_setup****has\_teardown****id**

Root TRY/EXCEPT id is always None.

**kname**

Deprecated.

**libname**

Deprecated.

**message**

Deprecated.

**name**

Deprecated.

**not\_run**

True when *status* is 'NOT RUN', False otherwise.

Setting to False value is ambiguous and raises an exception.

**parent****passed**

True when *status* is 'PASS', False otherwise.

```

repr_args = ()

skipped
    True when status is 'SKIP', False otherwise.

    Setting to False value is ambiguous and raises an exception.

tags
    Deprecated.

timeout
    Deprecated.

try_branch

type = 'TRY/EXCEPT ROOT'

visit (visitor)

class robot.result.model.Return(values=(), status='FAIL', starttime=None, endtime=None,
                                parent=None)
    Bases: robot.model.control.Return, robot.result.model.StatusMixin, robot.result.model.deprecation.DeprecatedAttributesMixin

    body_class
        alias of Body

    status

    starttime

    endtime

    body
        Child keywords and messages as a Body object.

        Typically empty. Only contains something if running RETURN has failed due to a syntax error or listeners
        have logged messages or executed keywords.

    args
        Deprecated.

    doc
        Deprecated.

BREAK = 'BREAK'

CONTINUE = 'CONTINUE'

ELSE = 'ELSE'

ELSE_IF = 'ELSE IF'

EXCEPT = 'EXCEPT'

FAIL = 'FAIL'

FINALLY = 'FINALLY'

FOR = 'FOR'

IF = 'IF'

IF_ELSE_ROOT = 'IF/ELSE ROOT'

ITERATION = 'ITERATION'

KEYWORD = 'KEYWORD'

```

**MESSAGE** = 'MESSAGE'

**NOT\_RUN** = 'NOT RUN'

**NOT\_SET** = 'NOT SET'

**PASS** = 'PASS'

**RETURN** = 'RETURN'

**SETUP** = 'SETUP'

**SKIP** = 'SKIP'

**TEARDOWN** = 'TEARDOWN'

**TRY** = 'TRY'

**TRY\_EXCEPT\_ROOT** = 'TRY/EXCEPT ROOT'

**WHILE** = 'WHILE'

**assign**

Deprecated.

**config** (*\*\*attributes*)

Configure model object with given attributes.

`obj.config(name='Example', doc='Something')` is equivalent to setting `obj.name = 'Example'` and `obj.doc = 'Something'`.

New in Robot Framework 4.0.

**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.

**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.

**elapsedtime**

Total execution time in milliseconds.

**failed**

True when [`status`](#) is 'FAIL', False otherwise.

**has\_setup**

**has\_teardown**

**id**

Item id in format like `s1-t3-k1`.

See [`TestSuite.id`](#) for more information.

**kwname**  
Deprecated.

**libname**  
Deprecated.

**message**  
Deprecated.

**name**  
Deprecated.

**not\_run**  
True when *status* is 'NOT RUN', False otherwise.  
  
Setting to False value is ambiguous and raises an exception.

**parent**

**passed**  
True when *status* is 'PASS', False otherwise.

**repr\_args** = ('values',)

**skipped**  
True when *status* is 'SKIP', False otherwise.  
  
Setting to False value is ambiguous and raises an exception.

**tags**  
Deprecated.

**timeout**  
Deprecated.

**type** = 'RETURN'

**values**

**visit** (*visitor*)

```
class robot.result.model.Continue(status='FAIL', starttime=None, endtime=None, parent=None)
    Bases: robot.model.control.Continue, robot.result.model.StatusMixin, robot.result.model.deprecation.DeprecatedAttributesMixin
```

**body\_class**  
alias of *Body*

**status**

**starttime**

**endtime**

**body**  
Child keywords and messages as a *Body* object.  
  
Typically empty. Only contains something if running CONTINUE has failed due to a syntax error or listeners have logged messages or executed keywords.

**args**  
Deprecated.

**doc**  
Deprecated.

```
BREAK = 'BREAK'
CONTINUE = 'CONTINUE'
ELSE = 'ELSE'
ELSE_IF = 'ELSE IF'
EXCEPT = 'EXCEPT'
FAIL = 'FAIL'
FINALLY = 'FINALLY'
FOR = 'FOR'
IF = 'IF'
IF_ELSE_ROOT = 'IF/ELSE ROOT'
ITERATION = 'ITERATION'
KEYWORD = 'KEYWORD'
MESSAGE = 'MESSAGE'
NOT_RUN = 'NOT RUN'
NOT_SET = 'NOT SET'
PASS = 'PASS'
RETURN = 'RETURN'
SETUP = 'SETUP'
SKIP = 'SKIP'
TEARDOWN = 'TEARDOWN'
TRY = 'TRY'
TRY_EXCEPT_ROOT = 'TRY/EXCEPT ROOT'
WHILE = 'WHILE'

assign
    Deprecated.

config (**attributes)
    Configure model object with given attributes.

    obj.config(name='Example', doc='Something') is equivalent to setting obj.name =
    'Example' and obj.doc = 'Something'.

    New in Robot Framework 4.0.

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.

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.

**`elapsedtime`**

Total execution time in milliseconds.

**`failed`**

True when `status` is 'FAIL', False otherwise.

**`has_setup`**

**`has_teardown`**

**`id`**

Item id in format like `s1-t3-k1`.

See `TestSuite.id` for more information.

**`kwname`**

Deprecated.

**`libname`**

Deprecated.

**`message`**

Deprecated.

**`name`**

Deprecated.

**`not_run`**

True when `status` is 'NOT RUN', False otherwise.

Setting to False value is ambiguous and raises an exception.

**`parent`**

**`passed`**

True when `status` is 'PASS', False otherwise.

**`repr_args = ()`**

**`skipped`**

True when `status` is 'SKIP', False otherwise.

Setting to False value is ambiguous and raises an exception.

**`tags`**

Deprecated.

**`timeout`**

Deprecated.

**`type = 'CONTINUE'`**

**`visit`** (*visitor*)

**class** `robot.result.model.Break` (*status='FAIL', starttime=None, endtime=None, parent=None*)

Bases: `robot.model.control.Break`, `robot.result.model.StatusMixin`, `robot.result.model.deprecation.DeprecatedAttributesMixin`

**`body_class`**

alias of `Body`

**status**

**starttime**

**endtime**

**body**

Child keywords and messages as a *Body* object.

Typically empty. Only contains something if running BREAK has failed due to a syntax error or listeners have logged messages or executed keywords.

**args**

Deprecated.

**doc**

Deprecated.

**BREAK** = 'BREAK'

**CONTINUE** = 'CONTINUE'

**ELSE** = 'ELSE'

**ELSE\_IF** = 'ELSE IF'

**EXCEPT** = 'EXCEPT'

**FAIL** = 'FAIL'

**FINALLY** = 'FINALLY'

**FOR** = 'FOR'

**IF** = 'IF'

**IF\_ELSE\_ROOT** = 'IF/ELSE ROOT'

**ITERATION** = 'ITERATION'

**KEYWORD** = 'KEYWORD'

**MESSAGE** = 'MESSAGE'

**NOT\_RUN** = 'NOT RUN'

**NOT\_SET** = 'NOT SET'

**PASS** = 'PASS'

**RETURN** = 'RETURN'

**SETUP** = 'SETUP'

**SKIP** = 'SKIP'

**TEARDOWN** = 'TEARDOWN'

**TRY** = 'TRY'

**TRY\_EXCEPT\_ROOT** = 'TRY/EXCEPT ROOT'

**WHILE** = 'WHILE'

**assign**

Deprecated.



**config** (\*\*attributes)

Configure model object with given attributes.

`obj.config(name='Example', doc='Something')` is equivalent to setting `obj.name = 'Example'` and `obj.doc = 'Something'`.

New in Robot Framework 4.0.

**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.

**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.

**elapsedtime**

Total execution time in milliseconds.

**failed**

True when [status](#) is 'FAIL', False otherwise.

**has\_setup****has\_teardown****id**

Item id in format like `s1-t3-k1`.

See [TestSuite.id](#) for more information.

**kwname**

Deprecated.

**libname**

Deprecated.

**message**

Deprecated.

**name**

Deprecated.

**not\_run**

True when [status](#) is 'NOT RUN', False otherwise.

Setting to False value is ambiguous and raises an exception.

**parent****passed**

True when [status](#) is 'PASS', False otherwise.

**repr\_args = ()**

**skipped**

True when *status* is 'SKIP', False otherwise.

Setting to False value is ambiguous and raises an exception.

**tags**

Deprecated.

**timeout**

Deprecated.

**type = 'BREAK'****visit** (*visitor*)

```
class robot.result.model.Keyword(kwname="", libname="", doc="", args=(), assign=(),
                                tags=(), timeout=None, type='KEYWORD', status='FAIL',
                                starttime=None, endtime=None, parent=None, source-
                                name=None)
```

Bases: *robot.model.keyword.Keyword*, *robot.result.model.StatusMixin*

Represents results of a single keyword.

See the base class for documentation of attributes not documented here.

**body\_class**

alias of *Body*

**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. PASS, FAIL, SKIP or NOT RUN.

**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.

**sourcename**

Original name of keyword with embedded arguments.

**body**

Child keywords and messages as a *Body* object.

**keywords**

Deprecated since Robot Framework 4.0.

Use *body* or *teardown* instead.

**messages**

Keyword's messages.

Starting from Robot Framework 4.0 this is a list generated from messages in *body*.

**children**

List of child keywords and messages in creation order.

Deprecated since Robot Framework 4.0. Use **:att:'body'** instead.

**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.

**BREAK** = 'BREAK'

**CONTINUE** = 'CONTINUE'

**ELSE** = 'ELSE'

**ELSE\_IF** = 'ELSE IF'

**EXCEPT** = 'EXCEPT'

**FAIL** = 'FAIL'

**FINALLY** = 'FINALLY'

**FOR** = 'FOR'

**IF** = 'IF'

**IF\_ELSE\_ROOT** = 'IF/ELSE ROOT'

**ITERATION** = 'ITERATION'

**KEYWORD** = 'KEYWORD'

**MESSAGE** = 'MESSAGE'

**NOT\_RUN** = 'NOT RUN'

**NOT\_SET** = 'NOT SET'

**PASS** = 'PASS'

**RETURN** = 'RETURN'

**SETUP** = 'SETUP'

**SKIP** = 'SKIP'

**TEARDOWN** = 'TEARDOWN'

**TRY** = 'TRY'

**TRY\_EXCEPT\_ROOT** = 'TRY/EXCEPT ROOT'

**WHILE** = 'WHILE'

**args****assign****config** (*\*\*attributes*)

Configure model object with given attributes.

`obj.config(name='Example', doc='Something')` is equivalent to setting `obj.name = 'Example'` and `obj.doc = 'Something'`.

New in Robot Framework 4.0.

**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.

**`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.

**`doc`**

**`elapsedtime`**

Total execution time in milliseconds.

**`failed`**

True when `status` is 'FAIL', False otherwise.

**`has_setup`**

**`has_teardown`**

Check does a keyword have a teardown without creating a teardown object.

A difference between using `if kw.has_teardown:` and `if kw.teardown:` is that accessing the `teardown` attribute creates a `Keyword` object representing a teardown even when the keyword actually does not have one. This typically does not matter, but with bigger suite structures having lot of keywords it can have a considerable effect on memory usage.

New in Robot Framework 4.1.2.

**`id`**

Item id in format like `s1-t3-k1`.

See `TestSuite.id` for more information.

**`not_run`**

True when `status` is 'NOT RUN', False otherwise.

Setting to False value is ambiguous and raises an exception.

**`parent`**

**`passed`**

True when `status` is 'PASS', False otherwise.

**`repr_args = ('name', 'args', 'assign')`**

**`skipped`**

True when `status` is 'SKIP', False otherwise.

Setting to False value is ambiguous and raises an exception.

**`tags`**

Keyword tags as a `Tags` object.

**`teardown`**

Keyword teardown as a `Keyword` object.

Teardown can be modified by setting attributes directly:

```
keyword.teardown.name = 'Example'
keyword.teardown.args = ('First', 'Second')
```

Alternatively the `config()` method can be used to set multiple attributes in one call:

```
keyword.teardown.config(name='Example', args=('First', 'Second'))
```

The easiest way to reset the whole teardown is setting it to `None`. It will automatically recreate the underlying `Keyword` object:

```
keyword.teardown = None
```

This attribute is a `Keyword` object also when a keyword has no teardown but in that case its truth value is `False`. If there is a need to just check does a keyword have a teardown, using the `has_teardown` attribute avoids creating the `Keyword` object and is thus more memory efficient.

New in Robot Framework 4.0. Earlier teardown was accessed like `keyword.keywords.teardown`. `has_teardown` is new in Robot Framework 4.1.2.

**timeout**

**type**

**visit** (*visitor*)

*Visitor interface* entry-point.

```
class robot.result.model.TestCase (name="", doc="", tags=None, timeout=None, lineno=None,
                                   status='FAIL', message="", starttime=None, end-
                                   time=None, parent=None)
```

Bases: `robot.model.testcase.TestCase`, `robot.result.model.StatusMixin`

Represents results of a single test case.

See the base class for documentation of attributes not documented here.

**body\_class**

alias of `Body`

**fixture\_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`.

**not\_run**

True when `status` is 'NOT RUN', False otherwise.

Setting to False value is ambiguous and raises an exception.

**critical**

**FAIL** = 'FAIL'

**NOT\_RUN** = 'NOT RUN'

**NOT\_SET** = 'NOT SET'

**PASS** = 'PASS'

**SKIP** = 'SKIP'

**body**

Test body as a [Body](#) object.

**config** (\*\*attributes)

Configure model object with given attributes.

`obj.config(name='Example', doc='Something')` is equivalent to setting `obj.name = 'Example'` and `obj.doc = 'Something'`.

New in Robot Framework 4.0.

**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.

**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.

**doc**

**elapsedtime**

Total execution time in milliseconds.

**failed**

True when [status](#) is 'FAIL', False otherwise.

**has\_setup**

Check does a suite have a setup without creating a setup object.

A difference between using `if test.has_setup:` and `if test.setup:` is that accessing the [setup](#) attribute creates a [Keyword](#) object representing the setup even when the test actually does not have one. This typically does not matter, but with bigger suite structures containing a huge amount of tests it can have an effect on memory usage.

New in Robot Framework 5.0.

**has\_teardown**

Check does a test have a teardown without creating a teardown object.

See [has\\_setup](#) for more information.

New in Robot Framework 5.0.

**id**

Test case id in format like `s1-t3`.

See [TestSuite.id](#) for more information.

**keywords**

Deprecated since Robot Framework 4.0

Use *body*, *setup* or *teardown* instead.

**lineno****longname**

Test name prefixed with the long name of the parent suite.

**name****parent****passed**

True when *status* is 'PASS', False otherwise.

**repr\_args = ('name',)**

**setup**

Test setup as a *Keyword* object.

This attribute is a *Keyword* object also when a test has no setup but in that case its truth value is False.

Setup can be modified by setting attributes directly:

```
test.setup.name = 'Example'
test.setup.args = ('First', 'Second')
```

Alternatively the *config()* method can be used to set multiple attributes in one call:

```
test.setup.config(name='Example', args=('First', 'Second'))
```

The easiest way to reset the whole setup is setting it to None. It will automatically recreate the underlying *Keyword* object:

```
test.setup = None
```

New in Robot Framework 4.0. Earlier setup was accessed like `test.keywords.setup`.

**skipped**

True when *status* is 'SKIP', False otherwise.

Setting to False value is ambiguous and raises an exception.

**source****tags**

Test tags as a *Tags* object.

**teardown**

Test teardown as a *Keyword* object.

See *setup* for more information.

**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, parent=None)
```

Bases: *robot.model.testsuite.TestSuite*, *robot.result.model.StatusMixin*

Represents results of a single test suite.

See the base class for documentation of attributes not documented here.

**test\_class**

alias of *TestCase*

**fixture\_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 test has failed but some have passed, False otherwise.

**failed**

True if any test has failed, False otherwise.

**skipped**

True if there are no passed or failed tests, False otherwise.

**FAIL = 'FAIL'**

**NOT\_RUN = 'NOT RUN'**

**NOT\_SET = 'NOT SET'**

**PASS = 'PASS'**

**SKIP = 'SKIP'**

**config** (*\*\*attributes*)

Configure model object with given attributes.

`obj.config(name='Example', doc='Something')` is equivalent to setting `obj.name = 'Example'` and `obj.doc = 'Something'`.

New in Robot Framework 4.0.

**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.

**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.

**doc**



**filter** (*included\_suites=None, included\_tests=None, included\_tags=None, excluded\_tags=None*)

Select test cases and remove others from this suite.

Parameters have the same semantics as `--suite`, `--test`, `--include`, and `--exclude` command line options. All of them can be given as a list of strings, or when selecting only one, as a single string.

Child suites that contain no tests after filtering are automatically removed.

Example:

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

### **has\_setup**

Check does a suite have a setup without creating a setup object.

A difference between using `if suite.has_setup:` and `if suite.setup:` is that accessing the `setup` attribute creates a *Keyword* object representing the setup even when the suite actually does not have one. This typically does not matter, but with bigger suite structures containing a huge amount of suites it can have some effect on memory usage.

New in Robot Framework 5.0.

### **has\_teardown**

Check does a suite have a teardown without creating a teardown object.

See *has\_setup* for more information.

New in Robot Framework 5.0.

### **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**

Deprecated since Robot Framework 4.0

Use *setup* or *teardown* instead.

### **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.

### **not\_run**

True when *status* is 'NOT RUN', False otherwise.

Setting to False value is ambiguous and raises an exception.

### **parent**

**remove\_empty\_suites** (*preserve\_direct\_children=False*)

Removes all child suites not containing any tests, recursively.

**repr\_args** = ('name',)

**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.

**setup**

Suite setup as a *Keyword* object.

This attribute is a *Keyword* object also when a suite has no setup but in that case its truth value is *False*.

Setup can be modified by setting attributes directly:

```
suite.setup.name = 'Example'
suite.setup.args = ('First', 'Second')
```

Alternatively the *config()* method can be used to set multiple attributes in one call:

```
suite.setup.config(name='Example', args=('First', 'Second'))
```

The easiest way to reset the whole setup is setting it to *None*. It will automatically recreate the underlying *Keyword* object:

```
suite.setup = None
```

New in Robot Framework 4.0. Earlier setup was accessed like *suite.keywords.setup*.

**source**

**suites**

Child suites as a *TestSuites* object.

**teardown**

Suite teardown as a *Keyword* object.

See *setup* for more information.

**test\_count**

Number of the tests in this suite, recursively.

**tests**

Tests as a *TestCases* object.

**visit** (*visitor*)

*Visitor interface* entry-point.

**status**

'PASS', 'FAIL' or 'SKIP' depending on test statuses.

- If any test has failed, status is 'FAIL'.
- If no test has failed but at least some test has passed, status is 'PASS'.
- If there are no failed or passed tests, status is 'SKIP'. This covers both the case when all tests have been skipped and when there are no tests.

**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.failed)
print(stats.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.

**remove\_keywords** (*how*)

Remove keywords based on the given condition.

**Parameters** *how* – What approach to use when removing keywords. Either ALL, PASSED, FOR, WUKS, or NAME:<pattern>.

For more information about the possible values see the documentation of the `--removekeywords` command line option.

**filter\_messages** (*log\_level='TRACE'*)

Remove log messages below the specified *log\_level*.

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

A shortcut to configure a suite using one method call.

Can only be used with the root test suite.

**Parameters** *options* – Passed to *SuiteConfigurer* that will then set suite attributes, call *filter()*, etc. as needed.

Example:

```
suite.configure(remove_keywords='PASSED',
               doc='Smoke test results.')
```

Not to be confused with *config()* method that suites, tests, and keywords have to make it possible to set multiple attributes in one call.

**handle\_suite\_teardown\_failures** ()

Internal usage only.

**suite\_teardown\_failed** (*error*)

Internal usage only.

**suite\_teardown\_skipped** (*message*)

Internal usage only.

**robot.result.modeldeprecation module**

`robot.result.modeldeprecation.deprecated` (*method*)

```
class robot.result.modeldeprecation.DeprecatedAttributesMixin
    Bases: object

    name
        Deprecated.

    kwname
        Deprecated.

    libname
        Deprecated.

    args
        Deprecated.

    assign
        Deprecated.

    tags
        Deprecated.

    timeout
        Deprecated.

    message
        Deprecated.
```

### 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,                               in-
                                                         clude_keywords=True,
                                                         flat-
                                                         tened_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. Although the option name has word “keyword”, it controls also including FOR and IF structures.
- **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 a suite starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

**visit\_test**(*test*)

Implements traversing through tests.

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

**end\_body\_item**(*item*)

Called, by default, when keywords, messages or control structures end.

More specific `end_keyword()`, `end_message()`, `:meth:'end_for'`, etc. can be implemented to visit only keywords, messages or specific control structures.

Default implementation does nothing.

**end\_break**(*break\_*)

Called when a BREAK element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_continue**(*continue\_*)

Called when a CONTINUE element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_for**(*for\_*)

Called when a FOR loop ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_for\_iteration**(*iteration*)

Called when a FOR loop iteration ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_if**(*if\_*)

Called when an IF/ELSE structure ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_if\_branch**(*branch*)

Called when an IF/ELSE branch ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_keyword**(*keyword*)

Called when a keyword ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_message** (*msg*)

Called when a message ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_return** (*return\_*)

Called when a RETURN element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_suite** (*suite*)

Called when a suite ends. Default implementation does nothing.

**end\_test** (*test*)

Called when a test ends. Default implementation does nothing.

**end\_try** (*try\_*)

Called when a TRY/EXCEPT structure ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_try\_branch** (*branch*)

Called when TRY, EXCEPT, ELSE and FINALLY branches end.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_while** (*while\_*)

Called when a WHILE loop ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_while\_iteration** (*iteration*)

Called when a WHILE loop iteration ends.

By default, calls `end_body_item()` which, by default, does nothing.

**start\_body\_item** (*item*)

Called, by default, when keywords, messages or control structures start.

More specific `start_keyword()`, `start_message()`, `:meth:'start_for`, etc. can be implemented to visit only keywords, messages or specific control structures.

Can return explicit `False` to stop visiting. Default implementation does nothing.

**start\_break** (*break\_*)

Called when a BREAK element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_continue** (*continue\_*)

Called when a CONTINUE element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_for** (*for\_*)

Called when a FOR loop starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_for\_iteration** (*iteration*)

Called when a FOR loop iteration starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_if** (*if\_*)

Called when an IF/ELSE structure starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_if\_branch** (*branch*)

Called when an IF/ELSE branch starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_keyword** (*keyword*)

Called when a keyword starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_message** (*msg*)

Called when a message starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_return** (*return\_*)

Called when a RETURN element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_test** (*test*)

Called when a test starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

**start\_try** (*try\_*)

Called when a TRY/EXCEPT structure starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_try\_branch** (*branch*)

Called when TRY, EXCEPT, ELSE or FINALLY branches start.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_while** (*while\_*)

Called when a WHILE loop starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_while\_iteration** (*iteration*)

Called when a WHILE loop iteration starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**visit\_break** (*break\_*)

Visits BREAK elements.

**visit\_continue** (*continue\_*)

Visits CONTINUE elements.

**visit\_for** (*for\_*)

Implements traversing through FOR loops.

Can be overridden to allow modifying the passed in `for_` without calling `start_for()` or `end_for()` nor visiting body.

**visit\_for\_iteration** (*iteration*)

Implements traversing through single FOR loop iteration.

This is only used with the result side model because on the running side there are no iterations.

Can be overridden to allow modifying the passed in `iteration` without calling `start_for_iteration()` or `end_for_iteration()` nor visiting body.

**visit\_if** (*if\_*)

Implements traversing through IF/ELSE structures.

Notice that `if_` does not have any data directly. Actual IF/ELSE branches are in its body and visited using `visit_if_branch()`.

Can be overridden to allow modifying the passed in `if_` without calling `start_if()` or `end_if()` nor visiting branches.

**visit\_if\_branch** (*branch*)

Implements traversing through single IF/ELSE branch.

Can be overridden to allow modifying the passed in `branch` without calling `start_if_branch()` or `end_if_branch()` nor visiting body.

**visit\_keyword** (*kw*)

Implements traversing through keywords.

Can be overridden to allow modifying the passed in `kw` without calling `start_keyword()` or `end_keyword()` nor visiting the body of the keyword

**visit\_message** (*msg*)

Implements visiting messages.

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

**visit\_return** (*return\_*)

Visits a RETURN elements.

**visit\_suite** (*suite*)

Implements traversing through suites.

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



**visit\_try** (*try\_*)

Implements traversing through TRY/EXCEPT structures.

This method is used with the TRY/EXCEPT root element. Actual TRY, EXCEPT, ELSE and FINALLY branches are visited separately using `visit_try_branch()`.

**visit\_try\_branch** (*branch*)

Visits individual TRY, EXCEPT, ELSE and FINALLY branches.

**visit\_while** (*while\_*)

Implements traversing through WHILE loops.

Can be overridden to allow modifying the passed in `while_` without calling `start_while()` or `end_while()` nor visiting body.

**visit\_while\_iteration** (*iteration*)

Implements traversing through single WHILE loop iteration.

This is only used with the result side model because on the running side there are no iterations.

Can be overridden to allow modifying the passed in `iteration` without calling `start_while_iteration()` or `end_while_iteration()` nor visiting body.

**robot.result.suiteteardownfailed module****class** `robot.result.suiteteardownfailed.SuiteTeardownFailureHandler`

Bases: `robot.model.visitor.SuiteVisitor`

**end\_suite** (*suite*)

Called when a suite ends. Default implementation does nothing.

**visit\_test** (*test*)

Implements traversing through tests.

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

**visit\_keyword** (*keyword*)

Implements traversing through keywords.

Can be overridden to allow modifying the passed in `kw` without calling `start_keyword()` or `end_keyword()` nor visiting the body of the keyword

**end\_body\_item** (*item*)

Called, by default, when keywords, messages or control structures end.

More specific `end_keyword()`, `end_message()`, `:meth:'end_for'`, etc. can be implemented to visit only keywords, messages or specific control structures.

Default implementation does nothing.

**end\_break** (*break\_*)

Called when a BREAK element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_continue** (*continue\_*)

Called when a CONTINUE element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_for** (*for\_*)

Called when a FOR loop ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_for\_iteration** (*iteration*)

Called when a FOR loop iteration ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_if** (*if\_*)

Called when an IF/ELSE structure ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_if\_branch** (*branch*)

Called when an IF/ELSE branch ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_keyword** (*keyword*)

Called when a keyword ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_message** (*msg*)

Called when a message ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_return** (*return\_*)

Called when a RETURN element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_test** (*test*)

Called when a test ends. Default implementation does nothing.

**end\_try** (*try\_*)

Called when a TRY/EXCEPT structure ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_try\_branch** (*branch*)

Called when TRY, EXCEPT, ELSE and FINALLY branches end.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_while** (*while\_*)

Called when a WHILE loop ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_while\_iteration** (*iteration*)

Called when a WHILE loop iteration ends.

By default, calls `end_body_item()` which, by default, does nothing.

**start\_body\_item** (*item*)

Called, by default, when keywords, messages or control structures start.

More specific `start_keyword()`, `start_message()`, `:meth:'start_for`, etc. can be implemented to visit only keywords, messages or specific control structures.

Can return explicit `False` to stop visiting. Default implementation does nothing.

**start\_break** (*break\_*)

Called when a BREAK element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_continue** (*continue\_*)

Called when a CONTINUE element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_for** (*for\_*)

Called when a FOR loop starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_for\_iteration** (*iteration*)

Called when a FOR loop iteration starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_if** (*if\_*)

Called when an IF/ELSE structure starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_if\_branch** (*branch*)

Called when an IF/ELSE branch starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_keyword** (*keyword*)

Called when a keyword starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_message** (*msg*)

Called when a message starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_return** (*return\_*)

Called when a RETURN element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_suite** (*suite*)

Called when a suite starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

**start\_test** (*test*)

Called when a test starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

**start\_try** (*try\_*)

Called when a TRY/EXCEPT structure starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_try\_branch** (*branch*)

Called when TRY, EXCEPT, ELSE or FINALLY branches start.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_while** (*while\_*)

Called when a WHILE loop starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_while\_iteration** (*iteration*)

Called when a WHILE loop iteration starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**visit\_break** (*break\_*)

Visits BREAK elements.

**visit\_continue** (*continue\_*)

Visits CONTINUE elements.

**visit\_for** (*for\_*)

Implements traversing through FOR loops.

Can be overridden to allow modifying the passed in `for_` without calling `start_for()` or `end_for()` nor visiting body.

**visit\_for\_iteration** (*iteration*)

Implements traversing through single FOR loop iteration.

This is only used with the result side model because on the running side there are no iterations.

Can be overridden to allow modifying the passed in `iteration` without calling `start_for_iteration()` or `end_for_iteration()` nor visiting body.

**visit\_if** (*if\_*)

Implements traversing through IF/ELSE structures.

Notice that `if_` does not have any data directly. Actual IF/ELSE branches are in its `body` and visited using `visit_if_branch()`.

Can be overridden to allow modifying the passed in `if_` without calling `start_if()` or `end_if()` nor visiting branches.

**visit\_if\_branch** (*branch*)

Implements traversing through single IF/ELSE branch.

Can be overridden to allow modifying the passed in `branch` without calling `start_if_branch()` or `end_if_branch()` nor visiting body.

**visit\_message**(*msg*)

Implements visiting messages.

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

**visit\_return**(*return\_*)

Visits a RETURN elements.

**visit\_suite**(*suite*)

Implements traversing through suites.

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

**visit\_try**(*try\_*)

Implements traversing through TRY/EXCEPT structures.

This method is used with the TRY/EXCEPT root element. Actual TRY, EXCEPT, ELSE and FINALLY branches are visited separately using `visit_try_branch()`.

**visit\_try\_branch**(*branch*)

Visits individual TRY, EXCEPT, ELSE and FINALLY branches.

**visit\_while**(*while\_*)

Implements traversing through WHILE loops.

Can be overridden to allow modifying the passed in `while_` without calling `start_while()` or `end_while()` nor visiting body.

**visit\_while\_iteration**(*iteration*)

Implements traversing through single WHILE loop iteration.

This is only used with the result side model because on the running side there are no iterations.

Can be overridden to allow modifying the passed in `iteration` without calling `start_while_iteration()` or `end_while_iteration()` nor visiting body.

```
class robot.result.suiteteardownfailed.SuiteTeardownFailed(message,
                                                         skipped=False)
```

Bases: `robot.model.visitor.SuiteVisitor`

**visit\_test**(*test*)

Implements traversing through tests.

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

**visit\_keyword**(*keyword*)

Implements traversing through keywords.

Can be overridden to allow modifying the passed in `kw` without calling `start_keyword()` or `end_keyword()` nor visiting the body of the keyword

**end\_body\_item**(*item*)

Called, by default, when keywords, messages or control structures end.

More specific `end_keyword()`, `end_message()`, `:meth:'end_for'`, etc. can be implemented to visit only keywords, messages or specific control structures.

Default implementation does nothing.

**end\_break** (*break\_*)

Called when a BREAK element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_continue** (*continue\_*)

Called when a CONTINUE element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_for** (*for\_*)

Called when a FOR loop ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_for\_iteration** (*iteration*)

Called when a FOR loop iteration ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_if** (*if\_*)

Called when an IF/ELSE structure ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_if\_branch** (*branch*)

Called when an IF/ELSE branch ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_keyword** (*keyword*)

Called when a keyword ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_message** (*msg*)

Called when a message ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_return** (*return\_*)

Called when a RETURN element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_suite** (*suite*)

Called when a suite ends. Default implementation does nothing.

**end\_test** (*test*)

Called when a test ends. Default implementation does nothing.

**end\_try** (*try\_*)

Called when a TRY/EXCEPT structure ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_try\_branch** (*branch*)

Called when TRY, EXCEPT, ELSE and FINALLY branches end.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_while** (*while\_*)

Called when a WHILE loop ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_while\_iteration** (*iteration*)

Called when a WHILE loop iteration ends.

By default, calls `end_body_item()` which, by default, does nothing.

**start\_body\_item** (*item*)

Called, by default, when keywords, messages or control structures start.

More specific `start_keyword()`, `start_message()`, `:meth:'start_for`, etc. can be implemented to visit only keywords, messages or specific control structures.

Can return explicit `False` to stop visiting. Default implementation does nothing.

**start\_break** (*break\_*)

Called when a BREAK element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_continue** (*continue\_*)

Called when a CONTINUE element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_for** (*for\_*)

Called when a FOR loop starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_for\_iteration** (*iteration*)

Called when a FOR loop iteration starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_if** (*if\_*)

Called when an IF/ELSE structure starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_if\_branch** (*branch*)

Called when an IF/ELSE branch starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_keyword** (*keyword*)

Called when a keyword starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_message** (*msg*)

Called when a message starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_return** (*return\_*)

Called when a RETURN element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_suite** (*suite*)

Called when a suite starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

**start\_test** (*test*)

Called when a test starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

**start\_try** (*try\_*)

Called when a TRY/EXCEPT structure starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_try\_branch** (*branch*)

Called when TRY, EXCEPT, ELSE or FINALLY branches start.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_while** (*while\_*)

Called when a WHILE loop starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_while\_iteration** (*iteration*)

Called when a WHILE loop iteration starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**visit\_break** (*break\_*)

Visits BREAK elements.

**visit\_continue** (*continue\_*)

Visits CONTINUE elements.

**visit\_for** (*for\_*)

Implements traversing through FOR loops.

Can be overridden to allow modifying the passed in `for_` without calling `start_for()` or `end_for()` nor visiting body.

**visit\_for\_iteration** (*iteration*)

Implements traversing through single FOR loop iteration.

This is only used with the result side model because on the running side there are no iterations.

Can be overridden to allow modifying the passed in `iteration` without calling `start_for_iteration()` or `end_for_iteration()` nor visiting body.



**visit\_if** (*if\_*)

Implements traversing through IF/ELSE structures.

Notice that *if\_* does not have any data directly. Actual IF/ELSE branches are in its body and visited using *visit\_if\_branch()*.

Can be overridden to allow modifying the passed in *if\_* without calling *start\_if()* or *end\_if()* nor visiting branches.

**visit\_if\_branch** (*branch*)

Implements traversing through single IF/ELSE branch.

Can be overridden to allow modifying the passed in *branch* without calling *start\_if\_branch()* or *end\_if\_branch()* nor visiting body.

**visit\_message** (*msg*)

Implements visiting messages.

Can be overridden to allow modifying the passed in *msg* without calling *start\_message()* or *end\_message()*.

**visit\_return** (*return\_*)

Visits a RETURN elements.

**visit\_suite** (*suite*)

Implements traversing through suites.

Can be overridden to allow modifying the passed in *suite* without calling *start\_suite()* or *end\_suite()* nor visiting child suites, tests or setup and teardown at all.

**visit\_try** (*try\_*)

Implements traversing through TRY/EXCEPT structures.

This method is used with the TRY/EXCEPT root element. Actual TRY, EXCEPT, ELSE and FINALLY branches are visited separately using *visit\_try\_branch()*.

**visit\_try\_branch** (*branch*)

Visits individual TRY, EXCEPT, ELSE and FINALLY branches.

**visit\_while** (*while\_*)

Implements traversing through WHILE loops.

Can be overridden to allow modifying the passed in *while\_* without calling *start\_while()* or *end\_while()* nor visiting body.

**visit\_while\_iteration** (*iteration*)

Implements traversing through single WHILE loop iteration.

This is only used with the result side model because on the running side there are no iterations.

Can be overridden to allow modifying the passed in *iteration* without calling *start\_while\_iteration()* or *end\_while\_iteration()* nor visiting body.

**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\_body\_item** (*item*)

Called, by default, when keywords, messages or control structures end.

More specific *end\_keyword()*, *end\_message()*, *:meth:'end\_for'*, etc. can be implemented to visit only keywords, messages or specific control structures.

Default implementation does nothing.

**end\_break** (*break\_*)

Called when a BREAK element ends.

By default, calls *end\_body\_item()* which, by default, does nothing.

**end\_continue** (*continue\_*)  
Called when a CONTINUE element ends.  
By default, calls `end_body_item()` which, by default, does nothing.

**end\_for** (*for\_*)  
Called when a FOR loop ends.  
By default, calls `end_body_item()` which, by default, does nothing.

**end\_for\_iteration** (*iteration*)  
Called when a FOR loop iteration ends.  
By default, calls `end_body_item()` which, by default, does nothing.

**end\_if** (*if\_*)  
Called when an IF/ELSE structure ends.  
By default, calls `end_body_item()` which, by default, does nothing.

**end\_if\_branch** (*branch*)  
Called when an IF/ELSE branch ends.  
By default, calls `end_body_item()` which, by default, does nothing.

**end\_keyword** (*keyword*)  
Called when a keyword ends.  
By default, calls `end_body_item()` which, by default, does nothing.

**end\_message** (*msg*)  
Called when a message ends.  
By default, calls `end_body_item()` which, by default, does nothing.

**end\_return** (*return\_*)  
Called when a RETURN element ends.  
By default, calls `end_body_item()` which, by default, does nothing.

**end\_suite** (*suite*)  
Called when a suite ends. Default implementation does nothing.

**end\_test** (*test*)  
Called when a test ends. Default implementation does nothing.

**end\_try** (*try\_*)  
Called when a TRY/EXCEPT structure ends.  
By default, calls `end_body_item()` which, by default, does nothing.

**end\_try\_branch** (*branch*)  
Called when TRY, EXCEPT, ELSE and FINALLY branches end.  
By default, calls `end_body_item()` which, by default, does nothing.

**end\_while** (*while\_*)  
Called when a WHILE loop ends.  
By default, calls `end_body_item()` which, by default, does nothing.

**end\_while\_iteration** (*iteration*)  
Called when a WHILE loop iteration ends.  
By default, calls `end_body_item()` which, by default, does nothing.

**start\_body\_item** (*item*)

Called, by default, when keywords, messages or control structures start.

More specific `start_keyword()`, `start_message()`, `:meth:'start_for`, etc. can be implemented to visit only keywords, messages or specific control structures.

Can return explicit `False` to stop visiting. Default implementation does nothing.

**start\_break** (*break\_*)

Called when a BREAK element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_continue** (*continue\_*)

Called when a CONTINUE element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_for** (*for\_*)

Called when a FOR loop starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_for\_iteration** (*iteration*)

Called when a FOR loop iteration starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_if** (*if\_*)

Called when an IF/ELSE structure starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_if\_branch** (*branch*)

Called when an IF/ELSE branch starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_keyword** (*keyword*)

Called when a keyword starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_message** (*msg*)

Called when a message starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_return** (*return\_*)

Called when a RETURN element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_suite** (*suite*)

Called when a suite starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

**start\_test** (*test*)

Called when a test starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

**start\_try** (*try\_*)

Called when a TRY/EXCEPT structure starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_try\_branch** (*branch*)

Called when TRY, EXCEPT, ELSE or FINALLY branches start.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_while** (*while\_*)

Called when a WHILE loop starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_while\_iteration** (*iteration*)

Called when a WHILE loop iteration starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**visit\_break** (*break\_*)

Visits BREAK elements.

**visit\_continue** (*continue\_*)

Visits CONTINUE elements.

**visit\_for** (*for\_*)

Implements traversing through FOR loops.

Can be overridden to allow modifying the passed in `for_` without calling `start_for()` or `end_for()` nor visiting body.

**visit\_for\_iteration** (*iteration*)

Implements traversing through single FOR loop iteration.

This is only used with the result side model because on the running side there are no iterations.

Can be overridden to allow modifying the passed in `iteration` without calling `start_for_iteration()` or `end_for_iteration()` nor visiting body.

**visit\_if** (*if\_*)

Implements traversing through IF/ELSE structures.

Notice that `if_` does not have any data directly. Actual IF/ELSE branches are in its `body` and visited using `visit_if_branch()`.

Can be overridden to allow modifying the passed in `if_` without calling `start_if()` or `end_if()` nor visiting branches.

**visit\_if\_branch** (*branch*)

Implements traversing through single IF/ELSE branch.

Can be overridden to allow modifying the passed in `branch` without calling `start_if_branch()` or `end_if_branch()` nor visiting body.

**visit\_keyword** (*kw*)

Implements traversing through keywords.

Can be overridden to allow modifying the passed in `kw` without calling `start_keyword()` or `end_keyword()` nor visiting the body of the keyword

**visit\_message** (*msg*)

Implements visiting messages.

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

**visit\_return** (*return\_*)

Visits a RETURN elements.

**visit\_suite** (*suite*)

Implements traversing through suites.

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

**visit\_test** (*test*)

Implements traversing through tests.

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

**visit\_try** (*try\_*)

Implements traversing through TRY/EXCEPT structures.

This method is used with the TRY/EXCEPT root element. Actual TRY, EXCEPT, ELSE and FINALLY branches are visited separately using `visit_try_branch()`.

**visit\_try\_branch** (*branch*)

Visits individual TRY, EXCEPT, ELSE and FINALLY branches.

**visit\_while** (*while\_*)

Implements traversing through WHILE loops.

Can be overridden to allow modifying the passed in `while_` without calling `start_while()` or `end_while()` nor visiting body.

**visit\_while\_iteration** (*iteration*)

Implements traversing through single WHILE loop iteration.

This is only used with the result side model because on the running side there are no iterations.

Can be overridden to allow modifying the passed in `iteration` without calling `start_while_iteration()` or `end_while_iteration()` nor visiting body.

**robot.result.xmlélémenthandlers module**

```

class robot.result.xmlélémenthandlers.XmlElementHandler(execution_result,
                                                         root_handler=None)
    Bases: object
    start(elem)
    end(elem)

class robot.result.xmlélémenthandlers.ElementHandler
    Bases: object
    element_handlers = {'arg': <robot.result.xmlélémenthandlers.ArgumentHandler object>,
    tag = None
    children = frozenset()
    classmethod register(handler)
    get_child_handler(tag)
    start(elem, result)
    end(elem, result)

class robot.result.xmlélémenthandlers.RootHandler
    Bases: robot.result.xmlélémenthandlers.ElementHandler
    children = frozenset({'robot'})
    element_handlers = {'arg': <robot.result.xmlélémenthandlers.ArgumentHandler object>,
    end(elem, result)
    get_child_handler(tag)
    classmethod register(handler)
    start(elem, result)
    tag = None

class robot.result.xmlélémenthandlers.RobotHandler
    Bases: robot.result.xmlélémenthandlers.ElementHandler
    tag = 'robot'
    children = frozenset({'suite', 'errors', 'statistics'})
    start(elem, result)
    element_handlers = {'arg': <robot.result.xmlélémenthandlers.ArgumentHandler object>,
    end(elem, result)
    get_child_handler(tag)
    classmethod register(handler)

class robot.result.xmlélémenthandlers.SuiteHandler
    Bases: robot.result.xmlélémenthandlers.ElementHandler
    tag = 'suite'
    children = frozenset({'meta', 'status', 'metadata', 'doc', 'kw', 'suite', 'test'})
    start(elem, result)

```

```
    get_child_handler(tag)
    element_handlers = {'arg': <robot.result.xml_element_handlers.ArgumentHandler object>,
                        end(elem, result)
    classmethod register(handler)

class robot.result.xml_element_handlers.TestHandler
    Bases: robot.result.xml_element_handlers.ElementHandler
    tag = 'test'
    children = frozenset({'status', 'tags', 'for', 'timeout', 'break', 'continue', 'doc',
                          start(elem, result)
    element_handlers = {'arg': <robot.result.xml_element_handlers.ArgumentHandler object>,
                        end(elem, result)
    get_child_handler(tag)
    classmethod register(handler)

class robot.result.xml_element_handlers.KeywordHandler
    Bases: robot.result.xml_element_handlers.ElementHandler
    tag = 'kw'
    children = frozenset({'status', 'tags', 'for', 'timeout', 'assign', 'var', 'break', 'd
                          start(elem, result)
    element_handlers = {'arg': <robot.result.xml_element_handlers.ArgumentHandler object>,
                        end(elem, result)
    get_child_handler(tag)
    classmethod register(handler)

class robot.result.xml_element_handlers.ForHandler
    Bases: robot.result.xml_element_handlers.ElementHandler
    tag = 'for'
    children = frozenset({'status', 'var', 'doc', 'msg', 'kw', 'value', 'iter'})
    start(elem, result)
    element_handlers = {'arg': <robot.result.xml_element_handlers.ArgumentHandler object>,
                        end(elem, result)
    get_child_handler(tag)
    classmethod register(handler)

class robot.result.xml_element_handlers.WhileHandler
    Bases: robot.result.xml_element_handlers.ElementHandler
    tag = 'while'
    children = frozenset({'status', 'doc', 'msg', 'kw', 'iter'})
    start(elem, result)
    element_handlers = {'arg': <robot.result.xml_element_handlers.ArgumentHandler object>,
```



```
    end (elem, result)

    get_child_handler (tag)

    classmethod register (handler)

class robot.result.xml_element_handlers.IterationHandler
    Bases: robot.result.xml_element_handlers.ElementHandler

    tag = 'iter'

    children = frozenset({'status', 'for', 'var', 'break', 'doc', 'while', 'kw', 'msg', 'c
    start (elem, result)

    element_handlers = {'arg': <robot.result.xml_element_handlers.ArgumentHandler object>,
    end (elem, result)

    get_child_handler (tag)

    classmethod register (handler)

class robot.result.xml_element_handlers.IfHandler
    Bases: robot.result.xml_element_handlers.ElementHandler

    tag = 'if'

    children = frozenset({'status', 'branch', 'doc', 'msg', 'kw'})

    start (elem, result)

    element_handlers = {'arg': <robot.result.xml_element_handlers.ArgumentHandler object>,
    end (elem, result)

    get_child_handler (tag)

    classmethod register (handler)

class robot.result.xml_element_handlers.BranchHandler
    Bases: robot.result.xml_element_handlers.ElementHandler

    tag = 'branch'

    children = frozenset({'status', 'for', 'break', 'continue', 'doc', 'while', 'kw', 'msg
    start (elem, result)

    element_handlers = {'arg': <robot.result.xml_element_handlers.ArgumentHandler object>,
    end (elem, result)

    get_child_handler (tag)

    classmethod register (handler)

class robot.result.xml_element_handlers.TryHandler
    Bases: robot.result.xml_element_handlers.ElementHandler

    tag = 'try'

    children = frozenset({'status', 'branch', 'doc', 'msg', 'kw'})

    start (elem, result)

    element_handlers = {'arg': <robot.result.xml_element_handlers.ArgumentHandler object>,
    end (elem, result)
```

```
    get_child_handler(tag)
    classmethod register(handler)
class robot.result.xml_element_handlers.PatternHandler
    Bases: robot.result.xml_element_handlers.ElementHandler
    tag = 'pattern'
    children = frozenset()
    end(elem, result)
    element_handlers = {'arg': <robot.result.xml_element_handlers.ArgumentHandler object>,
    get_child_handler(tag)
    classmethod register(handler)
    start(elem, result)
class robot.result.xml_element_handlers.ReturnHandler
    Bases: robot.result.xml_element_handlers.ElementHandler
    tag = 'return'
    children = frozenset({'msg', 'status', 'value', 'kw'})
    start(elem, result)
    element_handlers = {'arg': <robot.result.xml_element_handlers.ArgumentHandler object>,
    end(elem, result)
    get_child_handler(tag)
    classmethod register(handler)
class robot.result.xml_element_handlers.ContinueHandler
    Bases: robot.result.xml_element_handlers.ElementHandler
    tag = 'continue'
    children = frozenset({'msg', 'status', 'kw'})
    start(elem, result)
    element_handlers = {'arg': <robot.result.xml_element_handlers.ArgumentHandler object>,
    end(elem, result)
    get_child_handler(tag)
    classmethod register(handler)
class robot.result.xml_element_handlers.BreakHandler
    Bases: robot.result.xml_element_handlers.ElementHandler
    tag = 'break'
    children = frozenset({'msg', 'status', 'kw'})
    start(elem, result)
    element_handlers = {'arg': <robot.result.xml_element_handlers.ArgumentHandler object>,
    end(elem, result)
    get_child_handler(tag)
```

```
    classmethod register(handler)

class robot.result.xml_element_handlers.MessageHandler
    Bases: robot.result.xml_element_handlers.ElementHandler
    tag = 'msg'
    end(elem, result)
    children = frozenset()
    element_handlers = {'arg': <robot.result.xml_element_handlers.ArgumentHandler object>,
    get_child_handler(tag)
    classmethod register(handler)
    start(elem, result)

class robot.result.xml_element_handlers.StatusHandler(set_status=True)
    Bases: robot.result.xml_element_handlers.ElementHandler
    tag = 'status'
    end(elem, result)
    children = frozenset()
    element_handlers = {'arg': <robot.result.xml_element_handlers.ArgumentHandler object>,
    get_child_handler(tag)
    classmethod register(handler)
    start(elem, result)

class robot.result.xml_element_handlers.DocHandler
    Bases: robot.result.xml_element_handlers.ElementHandler
    tag = 'doc'
    end(elem, result)
    children = frozenset()
    element_handlers = {'arg': <robot.result.xml_element_handlers.ArgumentHandler object>,
    get_child_handler(tag)
    classmethod register(handler)
    start(elem, result)

class robot.result.xml_element_handlers.MetadataHandler
    Bases: robot.result.xml_element_handlers.ElementHandler
    tag = 'metadata'
    children = frozenset({'item'})
    element_handlers = {'arg': <robot.result.xml_element_handlers.ArgumentHandler object>,
    end(elem, result)
    get_child_handler(tag)
    classmethod register(handler)
    start(elem, result)
```

```
class robot.result.xml_element_handlers.MetadataItemHandler
    Bases: robot.result.xml_element_handlers.ElementHandler
    tag = 'item'
    end(elem, result)
    children = frozenset()
    element_handlers = {'arg': <robot.result.xml_element_handlers.ArgumentHandler object>,
    get_child_handler(tag)
    classmethod register(handler)
    start(elem, result)

class robot.result.xml_element_handlers.MetaHandler
    Bases: robot.result.xml_element_handlers.ElementHandler
    tag = 'meta'
    end(elem, result)
    children = frozenset()
    element_handlers = {'arg': <robot.result.xml_element_handlers.ArgumentHandler object>,
    get_child_handler(tag)
    classmethod register(handler)
    start(elem, result)

class robot.result.xml_element_handlers.TagsHandler
    Bases: robot.result.xml_element_handlers.ElementHandler
    tag = 'tags'
    children = frozenset({'tag'})
    element_handlers = {'arg': <robot.result.xml_element_handlers.ArgumentHandler object>,
    end(elem, result)
    get_child_handler(tag)
    classmethod register(handler)
    start(elem, result)

class robot.result.xml_element_handlers.TagHandler
    Bases: robot.result.xml_element_handlers.ElementHandler
    tag = 'tag'
    end(elem, result)
    children = frozenset()
    element_handlers = {'arg': <robot.result.xml_element_handlers.ArgumentHandler object>,
    get_child_handler(tag)
    classmethod register(handler)
    start(elem, result)

class robot.result.xml_element_handlers.TimeoutHandler
    Bases: robot.result.xml_element_handlers.ElementHandler
```

```

    tag = 'timeout'
    end(elem, result)
    children = frozenset()
    element_handlers = {'arg': <robot.result.xml_element_handlers.ArgumentHandler object>,
    get_child_handler(tag)
    classmethod register(handler)
    start(elem, result)

class robot.result.xml_element_handlers.AssignHandler
    Bases: robot.result.xml_element_handlers.ElementHandler
    tag = 'assign'
    children = frozenset({'var'})
    element_handlers = {'arg': <robot.result.xml_element_handlers.ArgumentHandler object>,
    end(elem, result)
    get_child_handler(tag)
    classmethod register(handler)
    start(elem, result)

class robot.result.xml_element_handlers.VarHandler
    Bases: robot.result.xml_element_handlers.ElementHandler
    tag = 'var'
    end(elem, result)
    children = frozenset()
    element_handlers = {'arg': <robot.result.xml_element_handlers.ArgumentHandler object>,
    get_child_handler(tag)
    classmethod register(handler)
    start(elem, result)

class robot.result.xml_element_handlers.ArgumentsHandler
    Bases: robot.result.xml_element_handlers.ElementHandler
    tag = 'arguments'
    children = frozenset({'arg'})
    element_handlers = {'arg': <robot.result.xml_element_handlers.ArgumentHandler object>,
    end(elem, result)
    get_child_handler(tag)
    classmethod register(handler)
    start(elem, result)

class robot.result.xml_element_handlers.ArgumentHandler
    Bases: robot.result.xml_element_handlers.ElementHandler
    tag = 'arg'

```

```
    end (elem, result)
    children = frozenset()
    element_handlers = {'arg': <robot.result.xml_elementhandlers.ArgumentHandler object>,
    get_child_handler (tag)
    classmethod register (handler)
    start (elem, result)
class robot.result.xml_elementhandlers.ValueHandler
    Bases: robot.result.xml_elementhandlers.ElementHandler
    tag = 'value'
    end (elem, result)
    children = frozenset()
    element_handlers = {'arg': <robot.result.xml_elementhandlers.ArgumentHandler object>,
    get_child_handler (tag)
    classmethod register (handler)
    start (elem, result)
class robot.result.xml_elementhandlers.ErrorsHandler
    Bases: robot.result.xml_elementhandlers.ElementHandler
    tag = 'errors'
    start (elem, result)
    get_child_handler (tag)
    children = frozenset()
    element_handlers = {'arg': <robot.result.xml_elementhandlers.ArgumentHandler object>,
    end (elem, result)
    classmethod register (handler)
class robot.result.xml_elementhandlers.ErrorMessageHandler
    Bases: robot.result.xml_elementhandlers.ElementHandler
    end (elem, result)
    children = frozenset()
    element_handlers = {'arg': <robot.result.xml_elementhandlers.ArgumentHandler object>,
    get_child_handler (tag)
    classmethod register (handler)
    start (elem, result)
    tag = None
class robot.result.xml_elementhandlers.StatisticsHandler
    Bases: robot.result.xml_elementhandlers.ElementHandler
    tag = 'statistics'
    get_child_handler (tag)
```

```

children = frozenset()
element_handlers = {'arg': <robot.result.xml_element_handlers.ArgumentHandler object>,
end(elem, result)
classmethod register(handler)
start(elem, result)

```

## robot.running package

Implements the core test execution logic.

The main public entry points of this package are of the following two classes:

- `TestSuiteBuilder` for creating executable test suites based on existing test case files and directories.
- `TestSuite` for creating an executable test suite structure programmatically.

It is recommended to import both of these classes via the `robot.api` package like in the examples below. Also `TestCase` and `Keyword` classes used internally by the `TestSuite` class are part of the public API. In those rare cases where these classes are needed directly, they can be imported from this package.

## Examples

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

```

*** Settings ***
Library      OperatingSystem

*** Test Cases ***
Should Activate Skynet
    [Tags]    smoke
    [Setup]    Set Environment Variable      SKYNET      activated
    Environment Variable Should Be Set      SKYNET

```

We can easily parse and create an executable test suite based on the above file using the `TestSuiteBuilder` class as follows:

```

from robot.api import TestSuiteBuilder

suite = TestSuiteBuilder().build('path/to/activate_skynet.robot')

```

That was easy. Let's next generate the same test suite from scratch using the `TestSuite` class:

```

from robot.api import TestSuite

suite = TestSuite('Activate Skynet')
suite.resource.imports.library('OperatingSystem')
test = suite.tests.create('Should Activate Skynet', tags=['smoke'])
test.setup.config(name='Set Environment Variable', args=['SKYNET', 'activated'])
test.body.create_keyword('Environment Variable Should Be Set', args=['SKYNET'])

```

Not that complicated either, especially considering the flexibility. Notice that the suite created based on the file could also be edited further using the same API.

Now that we have a test suite ready, let's *execute it* and verify that the returned `Result` object contains correct information:

```
result = suite.run(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
stats = result.suite.statistics
assert stats.total == 1 and stats.passed == 1 and stats.failed == 0
```

Running the suite generates a normal output XML file, unless it is disabled by using `output=None`. Generating log, report, and xUnit files based on the results is possible using the `ResultWriter` class:

```
from robot.api import ResultWriter

# Report and xUnit files can be generated based on the result object.
ResultWriter(result).write_results(report='skynet.html', log=None)
# Generating log files requires processing the earlier generated output XML.
ResultWriter('skynet.xml').write_results()
```

## Subpackages

### robot.running.arguments package

## Submodules

### robot.running.arguments.argumentconverter module

```
class robot.running.arguments.argumentconverter.ArgumentConverter (argspec,
                                                                    converters,
                                                                    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

```
class robot.running.arguments.argumentparser.PythonArgumentParser (type='Keyword',
                                                                    er-
                                                                    ror_reporter=None)

    Bases: robot.running.arguments.argumentparser._ArgumentParser

    parse (handler, name=None)

class robot.running.arguments.argumentparser.DynamicArgumentParser (type='Keyword',
                                                                    er-
                                                                    ror_reporter=None)

    Bases: robot.running.arguments.argumentparser._ArgumentSpecParser

    parse (argspec, name=None)

class robot.running.arguments.argumentparser.UserKeywordArgumentParser (type='Keyword',
                                                                    er-
                                                                    ror_reporter=None)

    Bases: robot.running.arguments.argumentparser._ArgumentSpecParser

    parse (argspec, name=None)
```

### robot.running.arguments.argumentresolver module

```
class robot.running.arguments.argumentresolver.ArgumentResolver (argspec,    re-
                                                                    solve_named=True,
                                                                    re-
                                                                    solve_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,    en-
                                                                    abled=False)

    Bases: object

    handle (positional, named)

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

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

**robot.running.arguments.argumentspec module**

```
class robot.running.arguments.argumentspec.ArgumentSpec (name=None,  
                                                    type='Keyword',    posi-  
                                                    tional_only=None,  posi-  
                                                    tional_or_named=None,  
                                                    var_positional=None,  
                                                    named_only=None,  
                                                    var_named=None,  
                                                    defaults=None,  
                                                    types=None)
```

Bases: object

**types**

**positional**

**minargs**

**maxargs**

**argument\_names**

**resolve** (*arguments,* *variables=None,* *converters=None,* *resolve\_named=True,* *re-*  
 *solve\_variables\_until=None,* *dict\_to\_kwargs=False*)

**convert** (*positional,* *named,* *converters=None,* *dry\_run=False*)

**map** (*positional,* *named,* *replace\_defaults=True*)

```
class robot.running.arguments.argumentspec.ArgInfo (kind, name="", types=<object ob-  
                                                    ject>, default=<object object>)
```

Bases: object

**NOTSET** = <object object>

**POSITIONAL\_ONLY** = 'POSITIONAL\_ONLY'

**POSITIONAL\_ONLY\_MARKER** = 'POSITIONAL\_ONLY\_MARKER'

**POSITIONAL\_OR\_NAMED** = 'POSITIONAL\_OR\_NAMED'

**VAR\_POSITIONAL** = 'VAR\_POSITIONAL'

**NAMED\_ONLY\_MARKER** = 'NAMED\_ONLY\_MARKER'

**NAMED\_ONLY** = 'NAMED\_ONLY'

**VAR\_NAMED** = 'VAR\_NAMED'

**types**

**required**

**types\_reprs**

**default\_repr**

**robot.running.arguments.argumentvalidator module**

```
class robot.running.arguments.argumentvalidator.ArgumentValidator (argspec)
```

Bases: object

**validate** (*positional, named, dryrun=False*)

### robot.running.arguments.customconverters module

```
class robot.running.arguments.customconverters.CustomArgumentConverters (converters)
    Bases: object
        classmethod from_dict (converters, error_reporter)
        get_converter_info (type_)
class robot.running.arguments.customconverters.ConverterInfo (type, converter,
                                                                value_types)
    Bases: object
    name
    doc
    classmethod for_converter (type_, converter)
```

### 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.typeconverters module

```
class robot.running.arguments.typeconverters.TypeConverter (used_type, custom_converters=None)
    Bases: object
    type = None
    type_name = None
    abc = None
    aliases = ()
    value_types = (<class 'str'>,)
    doc = None
    classmethod register (converter)
    classmethod converter_for (type_, custom_converters=None)
    classmethod handles (type_)
    convert (name, value, explicit_type=True, strict=True)
    no_conversion_needed (value)
class robot.running.arguments.typeconverters.EnumConverter (used_type, custom_converters=None)
    Bases: robot.running.arguments.typeconverters.TypeConverter
```

```
type
    alias of enum.Enum

type_name

value_types
    Built-in immutable sequence.

    If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized
    from iterable's items.

    If the argument is a tuple, the return value is the same object.

abc = None
aliases = ()

convert(name, value, explicit_type=True, strict=True)

classmethod converter_for(type_, custom_converters=None)

doc = None

classmethod handles(type_)

no_conversion_needed(value)

classmethod register(converter)

class robot.running.arguments.typeconverters.StringConverter(used_type,    cus-
                                                             tom_converters=None)
    Bases: robot.running.arguments.typeconverters.TypeConverter

    type
        alias of builtins.str

    type_name = 'string'

    aliases = ('string', 'str', 'unicode')

    value_types = (typing.Any,)

    abc = None

    convert(name, value, explicit_type=True, strict=True)

    classmethod converter_for(type_, custom_converters=None)

    doc = None

    classmethod handles(type_)

    no_conversion_needed(value)

    classmethod register(converter)

class robot.running.arguments.typeconverters.BooleanConverter(used_type,    cus-
                                                             tom_converters=None)
    Bases: robot.running.arguments.typeconverters.TypeConverter

    type
        alias of builtins.bool

    type_name = 'boolean'

    aliases = ('bool',)

    value_types = (<class 'str'>, <class 'int'>, <class 'float'>, <class 'NoneType'>)
```

```

    abc = None
    convert (name, value, explicit_type=True, strict=True)
    classmethod converter_for (type_, custom_converters=None)
    doc = None
    classmethod handles (type_)
    no_conversion_needed (value)
    classmethod register (converter)

class robot.running.arguments.typeconverters.IntegerConverter (used_type,    cus-
                                                                tom_converters=None)
    Bases: robot.running.arguments.typeconverters.TypeConverter
    type
        alias of builtins.int
    abc
        alias of numbers.Integral
    type_name = 'integer'
    aliases = ('int', 'long')
    value_types = (<class 'str'>, <class 'float'>)
    convert (name, value, explicit_type=True, strict=True)
    classmethod converter_for (type_, custom_converters=None)
    doc = None
    classmethod handles (type_)
    no_conversion_needed (value)
    classmethod register (converter)

class robot.running.arguments.typeconverters.FloatConverter (used_type,    cus-
                                                                tom_converters=None)
    Bases: robot.running.arguments.typeconverters.TypeConverter
    type
        alias of builtins.float
    abc
        alias of numbers.Real
    type_name = 'float'
    aliases = ('double',)
    value_types = (<class 'str'>, <class 'numbers.Real'>)
    convert (name, value, explicit_type=True, strict=True)
    classmethod converter_for (type_, custom_converters=None)
    doc = None
    classmethod handles (type_)
    no_conversion_needed (value)
    classmethod register (converter)

```

```
class robot.running.arguments.typeconverters.DecimalConverter(used_type, cus-  
                                                             tom_converters=None)  
    Bases: robot.running.arguments.typeconverters.TypeConverter  
    type  
        alias of decimal.Decimal  
    type_name = 'decimal'  
    value_types = (<class 'str'>, <class 'int'>, <class 'float'>)  
    abc = None  
    aliases = ()  
    convert (name, value, explicit_type=True, strict=True)  
    classmethod converter_for (type_, custom_converters=None)  
    doc = None  
    classmethod handles (type_)  
    no_conversion_needed (value)  
    classmethod register (converter)  
  
class robot.running.arguments.typeconverters.BytesConverter(used_type, cus-  
                                                             tom_converters=None)  
    Bases: robot.running.arguments.typeconverters.TypeConverter  
    type  
        alias of builtins.bytes  
    abc  
        alias of collections.abc.ByteString  
    type_name = 'bytes'  
    value_types = (<class 'str'>, <class 'bytearray'>)  
    aliases = ()  
    convert (name, value, explicit_type=True, strict=True)  
    classmethod converter_for (type_, custom_converters=None)  
    doc = None  
    classmethod handles (type_)  
    no_conversion_needed (value)  
    classmethod register (converter)  
  
class robot.running.arguments.typeconverters.ByteArrayConverter(used_type,  
                                                                cus-  
                                                                tom_converters=None)  
    Bases: robot.running.arguments.typeconverters.TypeConverter  
    type  
        alias of builtins.bytearray  
    type_name = 'bytearray'  
    value_types = (<class 'str'>, <class 'bytes'>)  
    abc = None
```

```

    aliases = ()
    convert (name, value, explicit_type=True, strict=True)
    classmethod converter_for (type_, custom_converters=None)
    doc = None
    classmethod handles (type_)
    no_conversion_needed (value)
    classmethod register (converter)
class robot.running.arguments.typeconverters.DateTimeConverter (used_type, cus-
                                                                    tom_converters=None)
    Bases: robot.running.arguments.typeconverters.TypeConverter
    type
        alias of datetime.datetime
    type_name = 'datetime'
    value_types = (<class 'str'>, <class 'int'>, <class 'float'>)
    abc = None
    aliases = ()
    convert (name, value, explicit_type=True, strict=True)
    classmethod converter_for (type_, custom_converters=None)
    doc = None
    classmethod handles (type_)
    no_conversion_needed (value)
    classmethod register (converter)
class robot.running.arguments.typeconverters.DateConverter (used_type,      cus-
                                                                tom_converters=None)
    Bases: robot.running.arguments.typeconverters.TypeConverter
    type
        alias of datetime.date
    type_name = 'date'
    abc = None
    aliases = ()
    convert (name, value, explicit_type=True, strict=True)
    classmethod converter_for (type_, custom_converters=None)
    doc = None
    classmethod handles (type_)
    no_conversion_needed (value)
    classmethod register (converter)
    value_types = (<class 'str'>,)

```

```
class robot.running.arguments.typeconverters.TimeDeltaConverter (used_type,  
                                                                cus-  
                                                                tom_converters=None)  
  
    Bases: robot.running.arguments.typeconverters.TypeConverter  
  
    type  
        alias of datetime.timedelta  
  
    type_name = 'timedelta'  
  
    value_types = (<class 'str'>, <class 'int'>, <class 'float'>)  
  
    abc = None  
  
    aliases = ()  
  
    convert (name, value, explicit_type=True, strict=True)  
  
    classmethod converter_for (type_, custom_converters=None)  
  
    doc = None  
  
    classmethod handles (type_)  
  
    no_conversion_needed (value)  
  
    classmethod register (converter)  
  
class robot.running.arguments.typeconverters.NoneConverter (used_type,      cus-  
                                                                tom_converters=None)  
  
    Bases: robot.running.arguments.typeconverters.TypeConverter  
  
    type  
        alias of builtins.NoneType  
  
    type_name = 'None'  
  
    classmethod handles (type_)  
  
    abc = None  
  
    aliases = ()  
  
    convert (name, value, explicit_type=True, strict=True)  
  
    classmethod converter_for (type_, custom_converters=None)  
  
    doc = None  
  
    no_conversion_needed (value)  
  
    classmethod register (converter)  
  
    value_types = (<class 'str'>,)  
  
class robot.running.arguments.typeconverters.ListConverter (used_type,      cus-  
                                                                tom_converters=None)  
  
    Bases: robot.running.arguments.typeconverters.TypeConverter  
  
    type  
        alias of builtins.list  
  
    type_name = 'list'  
  
    abc  
        alias of collections.abc.Sequence  
  
    value_types = (<class 'str'>, <class 'collections.abc.Sequence'>)
```



```

    no_conversion_needed (value)
    aliases = ()
    convert (name, value, explicit_type=True, strict=True)
    classmethod converter_for (type_, custom_converters=None)
    doc = None
    classmethod handles (type_)
    classmethod register (converter)
class robot.running.arguments.typeconverters.TupleConverter (used_type,      cus-
                                                                tom_converters=None)
    Bases: robot.running.arguments.typeconverters.TypeConverter
    type
        alias of builtins.tuple
    type_name = 'tuple'
    value_types = (<class 'str'>, <class 'collections.abc.Sequence'>)
    abc = None
    aliases = ()
    convert (name, value, explicit_type=True, strict=True)
    classmethod converter_for (type_, custom_converters=None)
    doc = None
    classmethod handles (type_)
    no_conversion_needed (value)
    classmethod register (converter)
class robot.running.arguments.typeconverters.DictionaryConverter (used_type,
                                                                    cus-
                                                                    tom_converters=None)
    Bases: robot.running.arguments.typeconverters.TypeConverter
    type
        alias of builtins.dict
    abc
        alias of collections.abc.Mapping
    type_name = 'dictionary'
    aliases = ('dict', 'map')
    value_types = (<class 'str'>, <class 'collections.abc.Mapping'>)
    convert (name, value, explicit_type=True, strict=True)
    classmethod converter_for (type_, custom_converters=None)
    doc = None
    classmethod handles (type_)
    no_conversion_needed (value)
    classmethod register (converter)

```

```
class robot.running.arguments.typeconverters.SetConverter (used_type,          cus-  
                                                    tom_converters=None)  
    Bases: robot.running.arguments.typeconverters.TypeConverter  
  
    type  
        alias of builtins.set  
  
    abc  
        alias of collections.abc.Set  
  
    type_name = 'set'  
    value_types = (<class 'str'>, <class 'collections.abc.Container'>)  
    aliases = ()  
    convert (name, value, explicit_type=True, strict=True)  
    classmethod converter_for (type_, custom_converters=None)  
    doc = None  
    classmethod handles (type_)  
    no_conversion_needed (value)  
    classmethod register (converter)  
  
class robot.running.arguments.typeconverters.FrozenSetConverter (used_type,  
                                                                    cus-  
                                                                    tom_converters=None)  
    Bases: robot.running.arguments.typeconverters.TypeConverter  
  
    type  
        alias of builtins.frozenset  
  
    type_name = 'frozenset'  
    value_types = (<class 'str'>, <class 'collections.abc.Container'>)  
    abc = None  
    aliases = ()  
    convert (name, value, explicit_type=True, strict=True)  
    classmethod converter_for (type_, custom_converters=None)  
    doc = None  
    classmethod handles (type_)  
    no_conversion_needed (value)  
    classmethod register (converter)  
  
class robot.running.arguments.typeconverters.CombinedConverter (union,          cus-  
                                                                    tom_converters)  
    Bases: robot.running.arguments.typeconverters.TypeConverter  
  
    type = typing.Union  
    type_name  
    classmethod handles (type_)  
    no_conversion_needed (value)  
    abc = None
```

```

    aliases = ()
    convert (name, value, explicit_type=True, strict=True)
    classmethod converter_for (type_, custom_converters=None)
    doc = None
    classmethod register (converter)
    value_types = (<class 'str'>,)
class robot.running.arguments.typeconverters.CustomConverter (used_type,    con-
                                                                verter_info)
    Bases: robot.running.arguments.typeconverters.TypeConverter
    type_name
    doc
    value_types
        Built-in immutable sequence.

        If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized
        from iterable's items.

        If the argument is a tuple, the return value is the same object.
    abc = None
    aliases = ()
    convert (name, value, explicit_type=True, strict=True)
    classmethod converter_for (type_, custom_converters=None)
    classmethod handles (type_)
    no_conversion_needed (value)
    classmethod register (converter)
    type = None

```

### 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`

Builder to construct `TestSuite` objects based on data on the disk.

The `build()` method constructs executable `TestSuite` objects based on test data files or directories. There are two main use cases for this API:

- Execute the created suite by using its `run()` method. The suite can be modified before execution if needed.
- Inspect the suite to see, for example, what tests it has or what tags tests have. This can be more convenient than using the lower level `parsing` APIs but does not allow saving modified data back to the disk.

Both modifying the suite and inspecting what data it contains are easiest done by using the `visitor` interface.

This class is part of the public API and should be imported via the `robot.api` package.

**Parameters**

- **include\_suites** – List of suite names to include. If `None` or an empty list, all suites are included. Same as using `-suite` on the command line.
- **included\_extensions** – List of extensions of files to parse. Same as `-extension`.
- **rpa** – Explicit test execution mode. `True` for RPA and `False` for test automation. By default mode is got from data file headers and possible conflicting headers cause an error. Same as `-rpa` or `-norpa`.
- **allow\_empty\_suite** – Specify is it an error if the built suite contains no tests. Same as `-runemptysuite`.
- **process\_curdir** – Control processing the special `${CURDIR}` variable. It is resolved already at parsing time by default, but that can be changed by giving this argument `False` value.

**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

    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
```

**teardown**  
**force\_tags**  
**timeout**

```
class robot.running.builder.testsettings.TestSettings (defaults)  
    Bases: object  
  
    setup  
    teardown  
    timeout  
    template  
    tags
```

### robot.running.builder.transformers module

```
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\_For** (*node*)

**visit\_While** (*node*)

**visit\_If** (*node*)

**visit\_Try** (*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*)

**visit\_ReturnStatement** (*node*)

**visit\_Continue** (*node*)

**visit\_Break** (*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\_ReturnStatement** (*node*)

**visit\_Continue** (*node*)

**visit\_Break** (*node*)

**visit\_For** (*node*)

**visit\_While** (*node*)

**visit\_If** (*node*)

**visit\_Try** (*node*)

**generic\_visit** (*node*)  
Called if no explicit visitor function exists for a node.

**visit** (*node*)  
Visit a node.

**class** robot.running.builder.transformers.**ForBuilder** (*parent*)

Bases: ast.NodeVisitor

**build** (*node*)

**visit\_KeywordCall** (*node*)

**visit\_TemplateArguments** (*node*)

**visit\_For** (*node*)

**visit\_While** (*node*)

**visit\_If** (*node*)

**visit\_Try** (*node*)

**visit\_ReturnStatement** (*node*)

**visit\_Continue** (*node*)

**visit\_Break** (*node*)



**generic\_visit** (*node*)  
Called if no explicit visitor function exists for a node.

**visit** (*node*)  
Visit a node.

**class** robot.running.builder.transformers.**IfBuilder** (*parent*)  
Bases: ast.NodeVisitor

**build** (*node*)

**visit\_KeywordCall** (*node*)

**visit\_TemplateArguments** (*node*)

**visit\_For** (*node*)

**visit\_While** (*node*)

**visit\_If** (*node*)

**visit\_Try** (*node*)

**visit\_ReturnStatement** (*node*)

**visit\_Continue** (*node*)

**visit\_Break** (*node*)

**generic\_visit** (*node*)  
Called if no explicit visitor function exists for a node.

**visit** (*node*)  
Visit a node.

**class** robot.running.builder.transformers.**TryBuilder** (*parent*)  
Bases: ast.NodeVisitor

**build** (*node*)

**visit\_For** (*node*)

**visit\_While** (*node*)

**visit\_If** (*node*)

**visit\_Try** (*node*)

**visit\_ReturnStatement** (*node*)

**visit\_Continue** (*node*)

**visit\_Break** (*node*)

**visit\_KeywordCall** (*node*)

**visit\_TemplateArguments** (*node*)

**generic\_visit** (*node*)  
Called if no explicit visitor function exists for a node.

**visit** (*node*)  
Visit a node.

**class** robot.running.builder.transformers.**WhileBuilder** (*parent*)  
Bases: ast.NodeVisitor

**build** (*node*)

```
visit_KeywordCall (node)
visit_TemplateArguments (node)
visit_For (node)
visit_While (node)
visit_If (node)
visit_Try (node)
visit_ReturnStatement (node)
visit_Break (node)
visit_Continue (node)
generic_visit (node)
    Called if no explicit visitor function exists for a node.
visit (node)
    Visit a node.
```

```
robot.running.builder.transformers.format_error (errors)
```

### 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.posix module

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

### robot.running.timeouts.windows module

```
class robot.running.timeouts.windows.Timeout(timeout, error)
    Bases: object
    execute(runnable)
```

## Submodules

### robot.running.bodyrunner module

```
class robot.running.bodyrunner.BodyRunner(context, run=True, templated=False)
    Bases: object
    run(body)
class robot.running.bodyrunner.KeywordRunner(context, run=True)
    Bases: object
    run(step, name=None)
robot.running.bodyrunner.ForRunner(context, flavor='IN', run=True, templated=False)
class robot.running.bodyrunner.ForInRunner(context, run=True, templated=False)
    Bases: object
    flavor = 'IN'
    run(data)
class robot.running.bodyrunner.ForInRangeRunner(context, run=True, templated=False)
    Bases: robot.running.bodyrunner.ForInRunner
    flavor = 'IN RANGE'
    run(data)
class robot.running.bodyrunner.ForInZipRunner(context, run=True, templated=False)
    Bases: robot.running.bodyrunner.ForInRunner
    flavor = 'IN ZIP'
    run(data)
class robot.running.bodyrunner.ForInEnumerateRunner(context, run=True, tem-
    plated=False)
    Bases: robot.running.bodyrunner.ForInRunner
```

```
    flavor = 'IN ENUMERATE'

    run(data)

class robot.running.bodyrunner.WhileRunner(context, run=True, templated=False)
    Bases: object

    run(data)

class robot.running.bodyrunner.IfRunner(context, run=True, templated=False)
    Bases: object

    run(data)

class robot.running.bodyrunner.TryRunner(context, run=True, templated=False)
    Bases: object

    run(data)

class robot.running.bodyrunner.WhileLimit
    Bases: object

    is_valid = True

    classmethod create(limit, variables)

    limit_exceeded()

class robot.running.bodyrunner.DurationLimit(max_time)
    Bases: robot.running.bodyrunner.WhileLimit

    classmethod create(limit, variables)

    is_valid = True

    limit_exceeded()

class robot.running.bodyrunner.IterationCountLimit(max_iterations)
    Bases: robot.running.bodyrunner.WhileLimit

    classmethod create(limit, variables)

    is_valid = True

    limit_exceeded()

class robot.running.bodyrunner.NoLimit
    Bases: robot.running.bodyrunner.WhileLimit

    classmethod create(limit, variables)

    is_valid = True

    limit_exceeded()

class robot.running.bodyrunner.InvalidLimit(reason)
    Bases: robot.running.bodyrunner.WhileLimit

    is_valid = False

    classmethod create(limit, variables)

    limit_exceeded()
```

### 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
class robot.running.dynamicmethods.GetKeywordSource (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=None, docgetter=None)
```

```
class robot.running.handlers.EmbeddedArgumentsHandler (name_regexp, orig_handler)
    Bases: object

    library
    matches (name)
    create_runner (name)
    resolve_arguments (args, variables=None)
```

### robot.running.handlerstore module

```
class robot.running.handlerstore.HandlerStore (source, source_type)
    Bases: object

    LIBRARY_TYPE = '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, run=True)
    dry_run (kw, context)
class robot.running.librarykeywordrunner.EmbeddedArgumentsRunner (handler,
                                                                    name)
    Bases: robot.running.librarykeywordrunner.LibraryKeywordRunner
    dry_run (kw, context)
    libname
    library
    longname
    run (kw, context, run=True)
class robot.running.librarykeywordrunner.RunKeywordRunner (handler,
                                                            de-
                                                            fault_dry_run_keywords=False)
    Bases: robot.running.librarykeywordrunner.LibraryKeywordRunner
    dry_run (kw, context)
    libname
    library
    longname
    run (kw, context, run=True)

```

### 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 = False
    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 = False
start_suite ()
```

## robot.running.model module

Module implementing test execution related model objects.

When tests are executed normally, these objects are created based on the test data on the file system by `TestSuiteBuilder`, but external tools can also create an executable test suite model structure directly. Regardless the approach to create it, the model is executed by calling `run()` method of the root test suite. See the [robot.running](#) package level documentation for more information and examples.

The most important classes defined in this module are `TestSuite`, `TestCase` and `Keyword`. When tests are executed, these objects can be inspected and modified by [pre-run modifiers](#) and [listeners](#). The aforementioned objects are considered stable, but other objects in this module may still be changed in the future major releases.

**class** `robot.running.model.Body` (*parent=None, items=None*)

Bases: `robot.model.body.Body`

**append** (*item*)

S.append(value) – append value to the end of the sequence

**break\_class**

alias of `Break`

**clear** () → None – remove all items from S

**continue\_class**

alias of `Continue`

**count** (*value*) → integer – return number of occurrences of value

**create**

**create\_break** (*\*args, \*\*kwargs*)

**create\_continue** (*\*args, \*\*kwargs*)

**create\_for** (*\*args, \*\*kwargs*)

**create\_if** (*\*args, \*\*kwargs*)

**create\_keyword** (*\*args, \*\*kwargs*)

**create\_message** (*\*args, \*\*kwargs*)

**create\_return** (*\*args, \*\*kwargs*)

**create\_try** (*\*args, \*\*kwargs*)

**create\_while** (*\*args, \*\*kwargs*)

**extend** (*items*)

S.extend(iterable) – extend sequence by appending elements from the iterable

**filter** (*keywords=None, messages=None, predicate=None*)

Filter body items based on type and/or custom predicate.

To include or exclude items based on types, give matching arguments `True` or `False` values. For example, to include only keywords, use `body.filter(keywords=True)` and to exclude messages use



`body.filter(messages=False)`. Including and excluding by types at the same time is not supported and filtering my messages is supported only if the `Body` object actually supports messages.

Custom `predicate` is a callable getting each body item as an argument that must return `True/False` depending on should the item be included or not.

Selected items are returned as a list and the original body is not modified.

It was earlier possible to filter also based on `FOR` and `IF` types. That support was removed in RF 5.0 because it was not considered useful in general and because adding support for all new control structures would have required extra work. To exclude all control structures, use `body.filter(keywords=True, messages=True)` and to only include them use `body.filter(keywords=False, messages=False)`. For more detailed filtering it is possible to use `predicate`.

**flatten()**

Return steps so that `IF` and `TRY` structures are flattened.

Basically the `IF/ELSE` and `TRY/EXCEPT` root elements are replaced with their branches. This is how they are shown in log files.

**for\_class**

alias of *For*

**if\_class**

alias of *If*

**index** (*value* [, *start* [, *stop* ]]) → integer – return first index of value.

Raises `ValueError` if the value is not present.

Supporting `start` and `stop` arguments is optional, but recommended.

**insert** (*index*, *item*)

`S.insert(index, value)` – insert value before index

**keyword\_class**

alias of *Keyword*

**message\_class = None**

**pop** ([*index*]) → item – remove and return item at index (default last).

Raise `IndexError` if list is empty or index is out of range.

**classmethod register** (*item\_class*)

Register a virtual subclass of an ABC.

Returns the subclass, to allow usage as a class decorator.

**remove** (*value*)

`S.remove(value)` – remove first occurrence of value. Raise `ValueError` if the value is not present.

**return\_class**

alias of *Return*

**reverse** ()

`S.reverse()` – reverse *IN PLACE*

**sort** ()

**try\_class**

alias of *Try*

**visit** (*visitor*)

**while\_class**  
alias of *While*

**class** robot.running.model.**Keyword**(name="", doc="", args=(), assign=(), tags=(),  
  timeout=None, type='KEYWORD', parent=None,  
  lineno=None)

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.

**lineno**

**source**

**run** (context, run=True, templated=None)

**BREAK** = 'BREAK'

**CONTINUE** = 'CONTINUE'

**ELSE** = 'ELSE'

**ELSE\_IF** = 'ELSE IF'

**EXCEPT** = 'EXCEPT'

**FINALLY** = 'FINALLY'

**FOR** = 'FOR'

**IF** = 'IF'

**IF\_ELSE\_ROOT** = 'IF/ELSE ROOT'

**ITERATION** = 'ITERATION'

**KEYWORD** = 'KEYWORD'

**MESSAGE** = 'MESSAGE'

**RETURN** = 'RETURN'

**SETUP** = 'SETUP'

**TEARDOWN** = 'TEARDOWN'

**TRY** = 'TRY'

**TRY\_EXCEPT\_ROOT** = 'TRY/EXCEPT ROOT'

**WHILE** = 'WHILE'

**args**

**assign**

**config** (\*\*attributes)

Configure model object with given attributes.

obj.config(name='Example', doc='Something') is equivalent to setting obj.name = 'Example' and obj.doc = 'Something'.

New in Robot Framework 4.0.

**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.

**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.

**doc**

**has\_setup**

**has\_teardown**

Check does a keyword have a teardown without creating a teardown object.

A difference between using `if kw.has_teardown:` and `if kw.teardown:` is that accessing the `teardown` attribute creates a *Keyword* object representing a teardown even when the keyword actually does not have one. This typically does not matter, but with bigger suite structures having lot of keywords it can have a considerable effect on memory usage.

New in Robot Framework 4.1.2.

**id**

Item id in format like `s1-t3-k1`.

See `TestSuite.id` for more information.

**name**

**parent**

**repr\_args** = ('name', 'args', 'assign')

**tags**

Keyword tags as a *Tags* object.

**teardown**

Keyword teardown as a *Keyword* object.

Teardown can be modified by setting attributes directly:

```
keyword.teardown.name = 'Example'
keyword.teardown.args = ('First', 'Second')
```

Alternatively the `config()` method can be used to set multiple attributes in one call:

```
keyword.teardown.config(name='Example', args=('First', 'Second'))
```

The easiest way to reset the whole teardown is setting it to `None`. It will automatically recreate the underlying *Keyword* object:

```
keyword.teardown = None
```

This attribute is a `Keyword` object also when a keyword has no teardown but in that case its truth value is `False`. If there is a need to just check does a keyword have a teardown, using the `has_teardown` attribute avoids creating the `Keyword` object and is thus more memory efficient.

New in Robot Framework 4.0. Earlier teardown was accessed like `keyword.keywords.teardown`. `has_teardown` is new in Robot Framework 4.1.2.

**timeout**

**type**

**visit** (*visitor*)

*Visitor interface* entry-point.

**class** `robot.running.model.For`(*variables*, *flavor*, *values*, *parent=None*, *lineno=None*, *error=None*)

Bases: `robot.model.control.For`

**body\_class**

alias of `Body`

**lineno**

**error**

**source**

**run** (*context*, *run=True*, *templated=False*)

**BREAK** = 'BREAK'

**CONTINUE** = 'CONTINUE'

**ELSE** = 'ELSE'

**ELSE\_IF** = 'ELSE IF'

**EXCEPT** = 'EXCEPT'

**FINALLY** = 'FINALLY'

**FOR** = 'FOR'

**IF** = 'IF'

**IF\_ELSE\_ROOT** = 'IF/ELSE ROOT'

**ITERATION** = 'ITERATION'

**KEYWORD** = 'KEYWORD'

**MESSAGE** = 'MESSAGE'

**RETURN** = 'RETURN'

**SETUP** = 'SETUP'

**TEARDOWN** = 'TEARDOWN'

**TRY** = 'TRY'

**TRY\_EXCEPT\_ROOT** = 'TRY/EXCEPT ROOT'

**WHILE** = 'WHILE'

**body**

**config** (\*\**attributes*)

Configure model object with given attributes.

`obj.config(name='Example', doc='Something')` is equivalent to setting `obj.name = 'Example'` and `obj.doc = 'Something'`.

New in Robot Framework 4.0.

**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.

**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.

**flavor**

**has\_setup**

**has\_teardown**

**id**

Item id in format like `s1-t3-k1`.

See [TestSuite.id](#) for more information.

**keywords**

Deprecated since Robot Framework 4.0. Use [body](#) instead.

**parent**

**repr\_args** = ('variables', 'flavor', 'values')

**type** = 'FOR'

**values**

**variables**

**visit** (*visitor*)

**class** `robot.running.model.While` (*condition=None, limit=None, parent=None, lineno=None, error=None*)

Bases: `robot.model.control.While`

**body\_class**

alias of [Body](#)

**lineno**

**error**

**source**

**run** (*context, run=True, templated=False*)

**BREAK** = 'BREAK'

```
CONTINUE = 'CONTINUE'
ELSE = 'ELSE'
ELSE_IF = 'ELSE IF'
EXCEPT = 'EXCEPT'
FINALLY = 'FINALLY'
FOR = 'FOR'
IF = 'IF'
IF_ELSE_ROOT = 'IF/ELSE ROOT'
ITERATION = 'ITERATION'
KEYWORD = 'KEYWORD'
MESSAGE = 'MESSAGE'
RETURN = 'RETURN'
SETUP = 'SETUP'
TEARDOWN = 'TEARDOWN'
TRY = 'TRY'
TRY_EXCEPT_ROOT = 'TRY/EXCEPT ROOT'
WHILE = 'WHILE'

body
condition

config (**attributes)
    Configure model object with given attributes.

    obj.config(name='Example', doc='Something') is equivalent to setting obj.name =
    'Example' and obj.doc = 'Something'.

    New in Robot Framework 4.0.

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.

deepcopy (**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.

has_setup
has_teardown
```

```

id
    Item id in format like s1-t3-k1.
    See TestSuite.id for more information.

limit

parent

repr_args = ('condition', 'limit')

type = 'WHILE'

visit (visitor)

class robot.running.model.IfBranch (type='IF', condition=None, parent=None, lineno=None)
    Bases: robot.model.control.IfBranch

    body_class
        alias of Body

    lineno

    source

    BREAK = 'BREAK'
    CONTINUE = 'CONTINUE'
    ELSE = 'ELSE'
    ELSE_IF = 'ELSE IF'
    EXCEPT = 'EXCEPT'
    FINALLY = 'FINALLY'
    FOR = 'FOR'
    IF = 'IF'
    IF_ELSE_ROOT = 'IF/ELSE ROOT'
    ITERATION = 'ITERATION'
    KEYWORD = 'KEYWORD'
    MESSAGE = 'MESSAGE'
    RETURN = 'RETURN'
    SETUP = 'SETUP'
    TEARDOWN = 'TEARDOWN'
    TRY = 'TRY'
    TRY_EXCEPT_ROOT = 'TRY/EXCEPT ROOT'
    WHILE = 'WHILE'

    body

    condition

    config (**attributes)
        Configure model object with given attributes.

        obj.config(name='Example', doc='Something') is equivalent to setting obj.name =
        'Example' and obj.doc = 'Something'.

```

New in Robot Framework 4.0.

**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.

**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.

**has\_setup**

**has\_teardown**

**id**

Branch id omits IF/ELSE root from the parent id part.

**parent**

**repr\_args** = ('type', 'condition')

**type**

**visit** (*visitor*)

**class** `robot.running.model.If` (*parent=None, lineno=None, error=None*)

Bases: [`robot.model.control.If`](#)

**branch\_class**

alias of [`IfBranch`](#)

**lineno**

**error**

**source**

**run** (*context, run=True, templated=False*)

**BREAK** = 'BREAK'

**CONTINUE** = 'CONTINUE'

**ELSE** = 'ELSE'

**ELSE\_IF** = 'ELSE IF'

**EXCEPT** = 'EXCEPT'

**FINALLY** = 'FINALLY'

**FOR** = 'FOR'

**IF** = 'IF'

**IF\_ELSE\_ROOT** = 'IF/ELSE ROOT'

**ITERATION** = 'ITERATION'



```

KEYWORD = 'KEYWORD'
MESSAGE = 'MESSAGE'
RETURN = 'RETURN'
SETUP = 'SETUP'
TEARDOWN = 'TEARDOWN'
TRY = 'TRY'
TRY_EXCEPT_ROOT = 'TRY/EXCEPT ROOT'
WHILE = 'WHILE'

body
branches_class
    alias of robot.model.body.Branches
config (**attributes)
    Configure model object with given attributes.

    obj.config(name='Example', doc='Something') is equivalent to setting obj.name =
    'Example' and obj.doc = 'Something'.

    New in Robot Framework 4.0.
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.
deepcopy (**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.

has_setup
has_teardown
id
    Root IF/ELSE id is always None.
parent
repr_args = ()
type = 'IF/ELSE ROOT'
visit (visitor)

class robot.running.model.TryBranch (type='TRY', patterns=(), pattern_type=None, vari-
    able=None, parent=None, lineno=None)
    Bases: robot.model.control.TryBranch

body_class
    alias of Body

```

```
lineno
source
BREAK = 'BREAK'
CONTINUE = 'CONTINUE'
ELSE = 'ELSE'
ELSE_IF = 'ELSE IF'
EXCEPT = 'EXCEPT'
FINALLY = 'FINALLY'
FOR = 'FOR'
IF = 'IF'
IF_ELSE_ROOT = 'IF/ELSE ROOT'
ITERATION = 'ITERATION'
KEYWORD = 'KEYWORD'
MESSAGE = 'MESSAGE'
RETURN = 'RETURN'
SETUP = 'SETUP'
TEARDOWN = 'TEARDOWN'
TRY = 'TRY'
TRY_EXCEPT_ROOT = 'TRY/EXCEPT ROOT'
WHILE = 'WHILE'
body
config (**attributes)
    Configure model object with given attributes.

    obj.config(name='Example', doc='Something') is equivalent to setting obj.name =
    'Example' and obj.doc = 'Something'.

    New in Robot Framework 4.0.
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.
deepcopy (**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.
has_setup
```

```

    has_teardown
    id
        Branch id omits TRY/EXCEPT root from the parent id part.
    parent
    pattern_type
    patterns
    repr_args = ('type', 'patterns', 'pattern_type', 'variable')
    type
    variable
    visit (visitor)
class robot.running.model.Try (parent=None, lineno=None, error=None)
    Bases: robot.model.control.Try
    branch_class
        alias of TryBranch
    lineno
    error
    source
    run (context, run=True, templated=False)
    BREAK = 'BREAK'
    CONTINUE = 'CONTINUE'
    ELSE = 'ELSE'
    ELSE_IF = 'ELSE IF'
    EXCEPT = 'EXCEPT'
    FINALLY = 'FINALLY'
    FOR = 'FOR'
    IF = 'IF'
    IF_ELSE_ROOT = 'IF/ELSE ROOT'
    ITERATION = 'ITERATION'
    KEYWORD = 'KEYWORD'
    MESSAGE = 'MESSAGE'
    RETURN = 'RETURN'
    SETUP = 'SETUP'
    TEARDOWN = 'TEARDOWN'
    TRY = 'TRY'
    TRY_EXCEPT_ROOT = 'TRY/EXCEPT ROOT'
    WHILE = 'WHILE'
    body

```

**branches\_class**

alias of `robot.model.body.Branches`

**config** (*\*\*attributes*)

Configure model object with given attributes.

`obj.config(name='Example', doc='Something')` is equivalent to setting `obj.name = 'Example'` and `obj.doc = 'Something'`.

New in Robot Framework 4.0.

**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.

**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.

**else\_branch****except\_branches****finally\_branch****has\_setup****has\_teardown****id**

Root TRY/EXCEPT id is always None.

**parent****repr\_args** = ()**try\_branch****type** = 'TRY/EXCEPT ROOT'**visit** (*visitor*)

**class** `robot.running.model.Return` (*values=()*, *parent=None*, *lineno=None*, *error=None*)

Bases: `robot.model.control.Return`

**lineno****error****source****run** (*context*, *run=True*, *templated=False*)**BREAK** = 'BREAK'**CONTINUE** = 'CONTINUE'**ELSE** = 'ELSE'

```

ELSE_IF = 'ELSE IF'
EXCEPT = 'EXCEPT'
FINALLY = 'FINALLY'
FOR = 'FOR'
IF = 'IF'
IF_ELSE_ROOT = 'IF/ELSE ROOT'
ITERATION = 'ITERATION'
KEYWORD = 'KEYWORD'
MESSAGE = 'MESSAGE'
RETURN = 'RETURN'
SETUP = 'SETUP'
TEARDOWN = 'TEARDOWN'
TRY = 'TRY'
TRY_EXCEPT_ROOT = 'TRY/EXCEPT ROOT'
WHILE = 'WHILE'

config (**attributes)
    Configure model object with given attributes.

    obj.config(name='Example', doc='Something') is equivalent to setting obj.name =
    'Example' and obj.doc = 'Something'.

    New in Robot Framework 4.0.

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.

deepcopy (**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.

has_setup
has_teardown
id
    Item id in format like s1-t3-k1.

    See TestSuite.id for more information.

parent
repr_args = ('values',)

```

```
type = 'RETURN'
values
visit (visitor)

class robot.running.model.Continue (parent=None, lineno=None, error=None)
    Bases: robot.model.control.Continue

    lineno
    error
    source
    run (context, run=True, templated=False)
    BREAK = 'BREAK'
    CONTINUE = 'CONTINUE'
    ELSE = 'ELSE'
    ELSE_IF = 'ELSE IF'
    EXCEPT = 'EXCEPT'
    FINALLY = 'FINALLY'
    FOR = 'FOR'
    IF = 'IF'
    IF_ELSE_ROOT = 'IF/ELSE ROOT'
    ITERATION = 'ITERATION'
    KEYWORD = 'KEYWORD'
    MESSAGE = 'MESSAGE'
    RETURN = 'RETURN'
    SETUP = 'SETUP'
    TEARDOWN = 'TEARDOWN'
    TRY = 'TRY'
    TRY_EXCEPT_ROOT = 'TRY/EXCEPT ROOT'
    WHILE = 'WHILE'

    config (**attributes)
        Configure model object with given attributes.

        obj.config(name='Example', doc='Something') is equivalent to setting obj.name =
        'Example' and obj.doc = 'Something'.

        New in Robot Framework 4.0.

    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.
```

**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.

**has\_setup**

**has\_teardown**

**id**

Item id in format like s1-t3-k1.

See `TestSuite.id` for more information.

**parent**

**repr\_args** = ()

**type** = 'CONTINUE'

**visit** (*visitor*)

**class** `robot.running.model.Break` (*parent=None, lineno=None, error=None*)

Bases: `robot.model.control.Break`

**lineno**

**error**

**source**

**run** (*context, run=True, templated=False*)

**BREAK** = 'BREAK'

**CONTINUE** = 'CONTINUE'

**ELSE** = 'ELSE'

**ELSE\_IF** = 'ELSE IF'

**EXCEPT** = 'EXCEPT'

**FINALLY** = 'FINALLY'

**FOR** = 'FOR'

**IF** = 'IF'

**IF\_ELSE\_ROOT** = 'IF/ELSE ROOT'

**ITERATION** = 'ITERATION'

**KEYWORD** = 'KEYWORD'

**MESSAGE** = 'MESSAGE'

**RETURN** = 'RETURN'

**SETUP** = 'SETUP'

**TEARDOWN** = 'TEARDOWN'

**TRY** = 'TRY'

**TRY\_EXCEPT\_ROOT** = 'TRY/EXCEPT ROOT'

**WHILE** = 'WHILE'

**config** (\*\*attributes)

Configure model object with given attributes.

`obj.config(name='Example', doc='Something')` is equivalent to setting `obj.name = 'Example'` and `obj.doc = 'Something'`.

New in Robot Framework 4.0.

**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.

**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.

**has\_setup**

**has\_teardown**

**id**

Item id in format like `s1-t3-k1`.

See `TestSuite.id` for more information.

**parent**

**repr\_args** = ()

**type** = 'BREAK'

**visit** (visitor)

**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.

**body\_class**

Internal usage only.

alias of `Body`

**fixture\_class**

Internal usage only.

alias of `Keyword`

**template**

**source**



**body**

Test body as a [Body](#) object.

**config** (*\*\*attributes*)

Configure model object with given attributes.

`obj.config(name='Example', doc='Something')` is equivalent to setting `obj.name = 'Example'` and `obj.doc = 'Something'`.

New in Robot Framework 4.0.

**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.

**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.

**doc****has\_setup**

Check does a suite have a setup without creating a setup object.

A difference between using `if test.has_setup:` and `if test.setup:` is that accessing the [setup](#) attribute creates a [Keyword](#) object representing the setup even when the test actually does not have one. This typically does not matter, but with bigger suite structures containing a huge amount of tests it can have an effect on memory usage.

New in Robot Framework 5.0.

**has\_teardown**

Check does a test have a teardown without creating a teardown object.

See [has\\_setup](#) for more information.

New in Robot Framework 5.0.

**id**

Test case id in format like `s1-t3`.

See [TestSuite.id](#) for more information.

**keywords**

Deprecated since Robot Framework 4.0

Use [body](#), [setup](#) or [teardown](#) instead.

**lineno****longname**

Test name prefixed with the long name of the parent suite.

**name****parent**

**repr\_args** = ('name',)

**setup**

Test setup as a *Keyword* object.

This attribute is a *Keyword* object also when a test has no setup but in that case its truth value is `False`.

Setup can be modified by setting attributes directly:

```
test.setup.name = 'Example'
test.setup.args = ('First', 'Second')
```

Alternatively the *config()* method can be used to set multiple attributes in one call:

```
test.setup.config(name='Example', args=('First', 'Second'))
```

The easiest way to reset the whole setup is setting it to `None`. It will automatically recreate the underlying *Keyword* object:

```
test.setup = None
```

New in Robot Framework 4.0. Earlier setup was accessed like `test.keywords.setup`.

**tags**

Test tags as a *Tags* object.

**teardown**

Test teardown as a *Keyword* object.

See *setup* for more information.

**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*

**fixture\_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 *robot.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:

```
suite.configure(included_tags=['smoke'],
               doc='Smoke test results.')
```

Not to be confused with *config()* method that suites, tests, and keywords have to make it possible to set multiple attributes in one call.

**randomize** (*suites=True*, *tests=True*, *seed=None*)

Randomizes the order of suites and/or tests, recursively.

#### Parameters

- **suites** – Boolean controlling should suites be randomized.
- **tests** – Boolean controlling should tests be randomized.
- **seed** – Random seed. Can be given if previous random order needs to be re-created. Seed value is always shown in logs and reports.

**run** (*settings=None*, *\*\*options*)

Executes the suite based based the given settings or options.

#### Parameters

- **settings** – *RobotSettings* object to configure test execution.
- **options** – Used to construct new *RobotSettings* object if settings are not given.

**Returns** *Result* object with information about executed suites and tests.

If options are used, their names are the same as long command line options except without hyphens. Some options are ignored (see below), but otherwise they have the same semantics as on the command line. Options that can be given on the command line multiple times can be passed as lists like *variable=['VAR1:value1', 'VAR2:value2']*. If such an option is used only once, it can be given also as a single string like *variable='VAR:value'*.

Additionally listener option allows passing object directly instead of listener name, e.g. *run('tests.robot', listener=Listener())*.

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

Only options related to the actual test execution have an effect. For example, options related to selecting or modifying test cases or suites (e.g. `--include`, `--name`, `--prerunmodifier`) or creating logs and reports are silently ignored. The output XML generated as part of the execution can be configured, though. This includes disabling it with `output=None`.

Example:

```
stdout = StringIO()
result = suite.run(variable='EXAMPLE:value',
                   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.

#### **config** (\*\*attributes)

Configure model object with given attributes.

`obj.config(name='Example', doc='Something')` is equivalent to setting `obj.name = 'Example'` and `obj.doc = 'Something'`.

New in Robot Framework 4.0.

#### **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.

#### **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.

#### **doc**

#### **filter** (included\_suites=None, included\_tests=None, included\_tags=None, excluded\_tags=None)

Select test cases and remove others from this suite.

Parameters have the same semantics as `--suite`, `--test`, `--include`, and `--exclude` command line options. All of them can be given as a list of strings, or when selecting only one, as a single string.

Child suites that contain no tests after filtering are automatically removed.

Example:

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

**has\_setup**

Check does a suite have a setup without creating a setup object.

A difference between using `if suite.has_setup:` and `if suite.setup:` is that accessing the `setup` attribute creates a *Keyword* object representing the setup even when the suite actually does not have one. This typically does not matter, but with bigger suite structures containing a huge amount of suites it can have some effect on memory usage.

New in Robot Framework 5.0.

**has\_teardown**

Check does a suite have a teardown without creating a teardown object.

See *has\_setup* for more information.

New in Robot Framework 5.0.

**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**

Deprecated since Robot Framework 4.0

Use *setup* or *teardown* instead.

**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.

**repr\_args** = ('name',)**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.

**setup**

Suite setup as a *Keyword* object.

This attribute is a `Keyword` object also when a suite has no setup but in that case its truth value is `False`.

Setup can be modified by setting attributes directly:

```
suite.setup.name = 'Example'
suite.setup.args = ('First', 'Second')
```

Alternatively the `config()` method can be used to set multiple attributes in one call:

```
suite.setup.config(name='Example', args=('First', 'Second'))
```

The easiest way to reset the whole setup is setting it to `None`. It will automatically recreate the underlying `Keyword` object:

```
suite.setup = None
```

New in Robot Framework 4.0. Earlier setup was accessed like `suite.keywords.setup`.

**source****suites**

Child suites as a `TestSuites` object.

**teardown**

Suite teardown as a `Keyword` object.

See `setup` for more information.

**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, time-out=None, lineno=None, parent=None, error=None*)

Bases: `object`

**body**

Child keywords as a `Body` object.

**keywords**

Deprecated since Robot Framework 4.0.

Use `body` or `teardown` instead.

**teardown**

```

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)
        S.append(value) – append value to the end of the sequence
    clear () → None – remove all items from S
    count (value) → integer – return number of occurrences of value
    create (*args, **kwargs)
    extend (items)
        S.extend(iterable) – extend sequence by appending elements from the iterable
    index (value[, start[, stop]]) → integer – return first index of value.
        Raises ValueError if the value is not present.
        Supporting start and stop arguments is optional, but recommended.
    insert (index, item)
        S.insert(index, value) – insert value before index
    pop ([index]) → item – remove and return item at index (default last).
        Raise IndexError if list is empty or index is out of range.
    remove (value)
        S.remove(value) – remove first occurrence of value. Raise ValueError if the value is not present.
    reverse ()
        S.reverse() – reverse IN PLACE
    sort ()
    visit (visitor)
    library (name, args=(), alias=None, lineno=None)
    resource (path, lineno=None)
    variables (path, args=(), lineno=None)

```

### robot.running.modelcombiner module

```

class robot.running.modelcombiner.ModelCombiner (data, result, **priority)
    Bases: object
    data
    result
    priority

```

**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, li-
                                                         braries, resources)
    Bases: object

    recommend_similar_keywords (name, message)
        Return keyword names similar to 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 ()
```



**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 a suite starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

**visit\_test** (*test*)

Implements traversing through tests.

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

**visit\_keyword** (*kw*)

Implements traversing through keywords.

Can be overridden to allow modifying the passed in `kw` without calling `start_keyword()` or `end_keyword()` nor visiting the body of the keyword

**end\_body\_item** (*item*)

Called, by default, when keywords, messages or control structures end.

More specific `end_keyword()`, `end_message()`, `:meth:'end_for'`, etc. can be implemented to visit only keywords, messages or specific control structures.

Default implementation does nothing.

**end\_break** (*break\_*)

Called when a BREAK element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_continue** (*continue\_*)

Called when a CONTINUE element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_for** (*for\_*)

Called when a FOR loop ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_for\_iteration** (*iteration*)

Called when a FOR loop iteration ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_if** (*if\_*)

Called when an IF/ELSE structure ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_if\_branch** (*branch*)

Called when an IF/ELSE branch ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_keyword** (*keyword*)

Called when a keyword ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_message** (*msg*)

Called when a message ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_return** (*return\_*)

Called when a RETURN element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_suite** (*suite*)

Called when a suite ends. Default implementation does nothing.

**end\_test** (*test*)

Called when a test ends. Default implementation does nothing.

**end\_try** (*try\_*)

Called when a TRY/EXCEPT structure ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_try\_branch** (*branch*)

Called when TRY, EXCEPT, ELSE and FINALLY branches end.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_while** (*while\_*)

Called when a WHILE loop ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_while\_iteration** (*iteration*)

Called when a WHILE loop iteration ends.

By default, calls `end_body_item()` which, by default, does nothing.

**start\_body\_item** (*item*)

Called, by default, when keywords, messages or control structures start.

More specific `start_keyword()`, `start_message()`, `:meth:'start_for`, etc. can be implemented to visit only keywords, messages or specific control structures.

Can return explicit `False` to stop visiting. Default implementation does nothing.

**start\_break** (*break\_*)

Called when a BREAK element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_continue** (*continue\_*)

Called when a CONTINUE element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_for** (*for\_*)

Called when a FOR loop starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_for\_iteration** (*iteration*)

Called when a FOR loop iteration starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_if** (*if\_*)

Called when an IF/ELSE structure starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_if\_branch** (*branch*)

Called when an IF/ELSE branch starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_keyword** (*keyword*)

Called when a keyword starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_message** (*msg*)

Called when a message starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_return** (*return\_*)

Called when a RETURN element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_test** (*test*)

Called when a test starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

**start\_try** (*try\_*)

Called when a TRY/EXCEPT structure starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_try\_branch** (*branch*)

Called when TRY, EXCEPT, ELSE or FINALLY branches start.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_while** (*while\_*)

Called when a WHILE loop starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_while\_iteration** (*iteration*)

Called when a WHILE loop iteration starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**visit\_break** (*break\_*)

Visits BREAK elements.

**visit\_continue** (*continue\_*)

Visits CONTINUE elements.

**visit\_for** (*for\_*)

Implements traversing through FOR loops.

Can be overridden to allow modifying the passed in `for_` without calling `start_for()` or `end_for()` nor visiting body.

**visit\_for\_iteration** (*iteration*)

Implements traversing through single FOR loop iteration.

This is only used with the result side model because on the running side there are no iterations.

Can be overridden to allow modifying the passed in `iteration` without calling `start_for_iteration()` or `end_for_iteration()` nor visiting body.

**visit\_if** (*if\_*)

Implements traversing through IF/ELSE structures.

Notice that `if_` does not have any data directly. Actual IF/ELSE branches are in its body and visited using `visit_if_branch()`.

Can be overridden to allow modifying the passed in `if_` without calling `start_if()` or `end_if()` nor visiting branches.

**visit\_if\_branch** (*branch*)

Implements traversing through single IF/ELSE branch.

Can be overridden to allow modifying the passed in `branch` without calling `start_if_branch()` or `end_if_branch()` nor visiting body.

**visit\_message** (*msg*)

Implements visiting messages.

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

**visit\_return** (*return\_*)

Visits a RETURN elements.

**visit\_suite** (*suite*)

Implements traversing through suites.

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

**visit\_try** (*try\_*)

Implements traversing through TRY/EXCEPT structures.

This method is used with the TRY/EXCEPT root element. Actual TRY, EXCEPT, ELSE and FINALLY branches are visited separately using `visit_try_branch()`.

**visit\_try\_branch** (*branch*)

Visits individual TRY, EXCEPT, ELSE and FINALLY branches.

**visit\_while** (*while\_*)

Implements traversing through WHILE loops.

Can be overridden to allow modifying the passed in *while\_* without calling *start\_while()* or *end\_while()* nor visiting body.

**visit\_while\_iteration** (*iteration*)

Implements traversing through single WHILE loop iteration.

This is only used with the result side model because on the running side there are no iterations.

Can be overridden to allow modifying the passed in *iteration* without calling *start\_while\_iteration()* or *end\_while\_iteration()* nor visiting body.

## robot.running.runkwregister module

## robot.running.signalhandler module

## robot.running.status module

**class** robot.running.status.Failure

Bases: object

**class** robot.running.status.Exit (*failure\_mode=False,* *error\_mode=False,*  
*skip\_teardown\_mode=False*)

Bases: object

**failure\_occurred** (*fatal=False*)

**error\_occurred** ()

**teardown\_allowed**

**class** robot.running.status.SuiteStatus (*parent=None,* *exit\_on\_failure=False,*  
*exit\_on\_error=False,*  
*skip\_teardown\_on\_exit=False*)

Bases: robot.running.status.\_ExecutionStatus

**error\_occurred** ()

**failed**

**failure\_occurred** ()

**message**

**passed**

**setup\_executed** (*error=None*)

**status**

**teardown\_allowed**

**teardown\_executed** (*error=None*)

**class** robot.running.status.TestStatus (*parent, test, skip\_on\_failure=None, rpa=False*)

Bases: robot.running.status.\_ExecutionStatus

**test\_failed** (*message=None, error=None*)

**test\_skipped** (*message*)

**skip\_on\_failure\_after\_tag\_changes**

```
error_occurred()
failed
failure_occurred()
message
passed
setup_executed(error=None)
status
teardown_allowed
teardown_executed(error=None)

class robot.running.status.TestMessage(status)
    Bases: robot.running.status._Message
    setup_message = 'Setup failed:\n%s'
    teardown_message = 'Teardown failed:\n%s'
    setup_skipped_message = '%s'
    teardown_skipped_message = '%s'
    also_teardown_message = '%s\n\nAlso teardown failed:\n%s'
    also_teardown_skip_message = 'Skipped in teardown:\n%s\n\nEarlier message:\n%s'
    exit_on_fatal_message = 'Test execution stopped due to a fatal error.'
    exit_on_failure_message = '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:\n%s'
    setup_skipped_message = 'Skipped in suite setup:\n%s'
    teardown_skipped_message = 'Skipped in suite teardown:\n%s'
    teardown_message = 'Suite teardown failed:\n%s'
    also_teardown_message = '%s\n\nAlso suite teardown failed:\n%s'
    also_teardown_skip_message = 'Skipped in suite teardown:\n%s\n\nEarlier message:\n%s'
    message

class robot.running.status.ParentMessage(status)
    Bases: robot.running.status.SuiteMessage
    setup_message = 'Parent suite setup failed:\n%s'
    setup_skipped_message = 'Skipped in parent suite setup:\n%s'
    teardown_skipped_message = 'Skipped in parent suite teardown:\n%s'
    teardown_message = 'Parent suite teardown failed:\n%s'
    also_teardown_message = '%s\n\nAlso parent suite teardown failed:\n%s'
```

```
also_teardown_skip_message = 'Skipped in suite teardown:\n%s\n\nEarlier message:\n%s'
message
```

### robot.running.statusreporter module

```
class robot.running.statusreporter.StatusReporter (data, result, context, run=True, sup-
                                                press=False)
    Bases: object
```

### robot.running.suiterunner module

```
class robot.running.suiterunner.SuiteRunner (output, settings)
    Bases: robot.model.visitor.SuiteVisitor

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

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

    visit_test (test)
        Implements traversing through tests.
        Can be overridden to allow modifying the passed in test without calling start_test() or
        end_test() nor visiting the body of the test.

    end_body_item (item)
        Called, by default, when keywords, messages or control structures end.
        More specific end_keyword(), end_message(), :meth:'end_for', etc. can be implemented to visit
        only keywords, messages or specific control structures.
        Default implementation does nothing.

    end_break (break_)
        Called when a BREAK element ends.
        By default, calls end_body_item() which, by default, does nothing.

    end_continue (continue_)
        Called when a CONTINUE element ends.
        By default, calls end_body_item() which, by default, does nothing.

    end_for (for_)
        Called when a FOR loop ends.
        By default, calls end_body_item() which, by default, does nothing.

    end_for_iteration (iteration)
        Called when a FOR loop iteration ends.
        By default, calls end_body_item() which, by default, does nothing.

    end_if (if_)
        Called when an IF/ELSE structure ends.
        By default, calls end_body_item() which, by default, does nothing.
```

**end\_if\_branch** (*branch*)

Called when an IF/ELSE branch ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_keyword** (*keyword*)

Called when a keyword ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_message** (*msg*)

Called when a message ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_return** (*return\_*)

Called when a RETURN element ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_test** (*test*)

Called when a test ends. Default implementation does nothing.

**end\_try** (*try\_*)

Called when a TRY/EXCEPT structure ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_try\_branch** (*branch*)

Called when TRY, EXCEPT, ELSE and FINALLY branches end.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_while** (*while\_*)

Called when a WHILE loop ends.

By default, calls `end_body_item()` which, by default, does nothing.

**end\_while\_iteration** (*iteration*)

Called when a WHILE loop iteration ends.

By default, calls `end_body_item()` which, by default, does nothing.

**start\_body\_item** (*item*)

Called, by default, when keywords, messages or control structures start.

More specific `start_keyword()`, `start_message()`, `:meth:'start_for`, etc. can be implemented to visit only keywords, messages or specific control structures.

Can return explicit `False` to stop visiting. Default implementation does nothing.

**start\_break** (*break\_*)

Called when a BREAK element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_continue** (*continue\_*)

Called when a CONTINUE element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.



**start\_for** (*for\_*)

Called when a FOR loop starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_for\_iteration** (*iteration*)

Called when a FOR loop iteration starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_if** (*if\_*)

Called when an IF/ELSE structure starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_if\_branch** (*branch*)

Called when an IF/ELSE branch starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_keyword** (*keyword*)

Called when a keyword starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_message** (*msg*)

Called when a message starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_return** (*return\_*)

Called when a RETURN element starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_test** (*test*)

Called when a test starts. Default implementation does nothing.

Can return explicit `False` to stop visiting.

**start\_try** (*try\_*)

Called when a TRY/EXCEPT structure starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_try\_branch** (*branch*)

Called when TRY, EXCEPT, ELSE or FINALLY branches start.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_while** (*while\_*)

Called when a WHILE loop starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**start\_while\_iteration** (*iteration*)

Called when a WHILE loop iteration starts.

By default, calls `start_body_item()` which, by default, does nothing.

Can return explicit `False` to stop visiting.

**visit\_break** (*break\_*)

Visits BREAK elements.

**visit\_continue** (*continue\_*)

Visits CONTINUE elements.

**visit\_for** (*for\_*)

Implements traversing through FOR loops.

Can be overridden to allow modifying the passed in `for_` without calling `start_for()` or `end_for()` nor visiting body.

**visit\_for\_iteration** (*iteration*)

Implements traversing through single FOR loop iteration.

This is only used with the result side model because on the running side there are no iterations.

Can be overridden to allow modifying the passed in `iteration` without calling `start_for_iteration()` or `end_for_iteration()` nor visiting body.

**visit\_if** (*if\_*)

Implements traversing through IF/ELSE structures.

Notice that `if_` does not have any data directly. Actual IF/ELSE branches are in its `body` and visited using `visit_if_branch()`.

Can be overridden to allow modifying the passed in `if_` without calling `start_if()` or `end_if()` nor visiting branches.

**visit\_if\_branch** (*branch*)

Implements traversing through single IF/ELSE branch.

Can be overridden to allow modifying the passed in `branch` without calling `start_if_branch()` or `end_if_branch()` nor visiting body.

**visit\_keyword** (*kw*)

Implements traversing through keywords.

Can be overridden to allow modifying the passed in `kw` without calling `start_keyword()` or `end_keyword()` nor visiting the body of the keyword

**visit\_message** (*msg*)

Implements visiting messages.

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

**visit\_return** (*return\_*)

Visits a RETURN elements.

**visit\_suite** (*suite*)

Implements traversing through suites.

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

**visit\_try** (*try\_*)

Implements traversing through TRY/EXCEPT structures.

This method is used with the TRY/EXCEPT root element. Actual TRY, EXCEPT, ELSE and FINALLY branches are visited separately using `visit_try_branch()`.

**visit\_try\_branch** (*branch*)

Visits individual TRY, EXCEPT, ELSE and FINALLY branches.

**visit\_while** (*while\_*)

Implements traversing through WHILE loops.

Can be overridden to allow modifying the passed in *while\_* without calling `start_while()` or `end_while()` nor visiting body.

**visit\_while\_iteration** (*iteration*)

Implements traversing through single WHILE loop iteration.

This is only used with the result side model because on the running side there are no iterations.

Can be overridden to allow modifying the passed in *iteration* without calling `start_while_iteration()` or `end_while_iteration()` nor visiting body.

**robot.running.testlibraries module**

```
robot.running.testlibraries.TestLibrary (name,          args=None,          variables=None,
                                           create_handlers=True,          logger=
                                           <robot.output.logger.Logger object>)
```

**robot.running.usererrorhandler module**

```
class robot.running.usererrorhandler.UserErrorHandler (error, name, libname=None,
                                                         source=None, lineno=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.
- **source** (*str*) – Path to the source file.
- **lineno** (*int*) – Line number of the failing keyword.

**longname**

**doc**

```
shortdoc
create_runner (name)
run (kw, context, run=True)
dry_run (kw, context, run=True)
```

### 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
    Return type robot.running.arguments.ArgumentSpec
    run (kw, context, run=True)
    dry_run (kw, context)

class robot.running.userkeywordrunner.EmbeddedArgumentsRunner (handler, name)
    Bases: robot.running.userkeywordrunner.UserKeywordRunner
    arguments
    Return type robot.running.arguments.ArgumentSpec
    dry_run (kw, context)
    libname
```

**longname**

**run** (*kw*, *context*, *run=True*)

## robot.utils package

Various generic utility functions and classes.

Utilities are mainly for internal usage, but external libraries and tools may find some of them useful. Utilities are generally stable, but absolute backwards compatibility between major versions is not guaranteed.

All utilities are exposed via the `robot.utils` package, and should be used either like:

```
from robot import utils

assert utils.Matcher('H?llo').match('Hillo')
```

or:

```
from robot.utils import Matcher

assert Matcher('H?llo').match('Hillo')
```

`robot.utils.read_rest_data` (*rstfile*)

`robot.utils.py2to3` (*cls*)

Deprecated since RF 5.0. Use Python 3 features directly instead.

`robot.utils.py3to2` (*cls*)

Deprecated since RF 5.0. Never done anything when used on Python 3.

## Submodules

### robot.utils.application module

```
class robot.utils.application.Application(usage, name=None, version=None,
                                         arg_limits=None, env_options=None, log-
                                         ger=None, **auto_options)
```

Bases: object

**main** (*arguments*, *\*\*options*)

**validate** (*options*, *arguments*)

**execute\_cli** (*cli\_arguments*, *exit=True*)

**console** (*msg*)

**parse\_arguments** (*cli\_args*)

Public interface for parsing command line arguments.

**Parameters** *cli\_args* – Command line arguments as a list

**Returns** options (dict), arguments (list)

**Raises** *Information* when `-help` or `-version` used

**Raises** *DataError* when parsing fails

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

```
class robot.utils.application.DefaultLogger
    Bases: object

    info (message)

    error (message)

    close ()
```

### robot.utils.argumentparser module

```
robot.utils.argumentparser.cmdline2list (args, escaping=False)
```

```
class robot.utils.argumentparser.ArgumentParser (usage,          name=None,          ver-
                                                sion=None, arg_limits=None, val-
                                                idator=None,    env_options=None,
                                                auto_help=True,  auto_version=True,
                                                auto_pythonpath='DEPRECATED',
                                                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`. This functionality was deprecated in Robot Framework 5.0.

–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 AssertionError occur outside.

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

unittest license:

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

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

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

#### Examples:

```
import unittest
from robot.utils.asserts import assert_equal

class MyTests(unittest.TestCase):

    def test_old_style(self):
        self.assertEqual(1, 2, 'my msg')
```

(continues on next page)

(continued from previous page)

```
def test_new_style(self):
    assert_equal(1, 2, 'my msg')
```

Example output:

```
FF
=====
FAIL: test_old_style (example.MyTests)
-----
Traceback (most recent call last):
  File "example.py", line 7, in test_old_style
    self.assertEqual(1, 2, 'my msg')
AssertionError: my msg
=====
FAIL: test_new_style (example.MyTests)
-----
Traceback (most recent call last):
  File "example.py", line 10, in test_new_style
    assert_equal(1, 2, 'my msg')
  File "/path/to/robot/utils/asserts.py", line 181, in assert_equal
    _report_inequality_failure(first, second, msg, values, '!=')
  File "/path/to/robot/utils/asserts.py", line 229, in _report_inequality_failure
    raise AssertionError(msg)
AssertionError: my msg: 1 != 2
-----
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=<function safe\_str>*)

Fail if given objects are unequal as determined by the '==' operator.

`robot.utils.asserts.assert_not_equal` (*first, second, msg=None, values=True, formatter=<function safe\_str>*)

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.

Python's *unicodedata* module was not used here because importing it took several seconds on Jython. That could possibly be changed now.

[1] <https://github.com/robotframework/robotframework/issues/604> [2] <https://github.com/robotframework/robotframework/issues/1096>

`robot.utils.charwidth.get_char_width` (*char*)

## robot.utils.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 libraries 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 library where there's need for multiple concurrent connections, processes, etc.

This class is used also outside the core framework by SeleniumLibrary, SSHLibrary, etc. Backwards compatibility is thus important when doing changes.

**current = None**

Current active connection.

**current\_index**

**register** (*connection*, *alias=None*)

Registers given connection with optional alias and returns its index.

Given connection is set to be the *current* connection.

If alias is given, it must be a string. Aliases are case and space insensitive.

The index of the first connection after initialization, and after *close\_all()* or *empty\_cache()*, is 1, second is 2, etc.

**switch** (*alias\_or\_index*)

Switches to the connection specified by the given alias or index.

Updates *current* and also returns its new value.

Alias is whatever was given to *register()* method and indices are returned by it. Index can be given either as an integer or as a string that can be converted to an integer. Raises an error if no connection with the given index or alias found.

**get\_connection** (*alias\_or\_index=None*)

Get the connection specified by the given alias or index..

If *alias\_or\_index* is None, returns the current connection if it is active, or raises an error if it is not.

Alias is whatever was given to *register()* method and indices are returned by it. Index can be given either as an integer or as a string that can be converted to an integer. Raises an error if no connection with the given index or alias found.

**close\_all** (*closer\_method='close'*)

Closes connections using given closer method and empties cache.

If simply calling the closer method is not adequate for closing connections, clients should close connections themselves and use *empty\_cache()* afterwards.

**empty\_cache** ()

Empties the connection cache.

Indexes of the new connections starts from 1 after this.

**resolve\_alias\_or\_index** (*alias\_or\_index*)

**class** robot.utils.connectioncache.NoConnection (*message*)

Bases: object

**raise\_error** ()

## robot.utils.dotdict module

**class** robot.utils.dotdict.DotDict (\*args, \*\*kws)

Bases: collections.OrderedDict

**clear** () → None. Remove all items from od.

**copy** () → a shallow copy of od

**fromkeys()**

Create a new ordered dictionary with keys from iterable and values set to value.

**get()**

Return the value for key if key is in the dictionary, else default.

**items()** → a set-like object providing a view on D's items

**keys()** → a set-like object providing a view on D's keys

**move\_to\_end()**

Move an existing element to the end (or beginning if last is false).

Raise KeyError if the element does not exist.

**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()**

Remove and return a (key, value) pair from the dictionary.

Pairs are returned in LIFO order if last is true or FIFO order if false.

**setdefault()**

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

**update(*[E]*, *\*\*F*)** → None. Update D from dict/iterable E and F.

If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

**values()** → an object providing a view on D's values

**robot.utils.encoding module**

`robot.utils.encoding.console_decode(string, encoding='UTF-8')`

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.

If *string* is already Unicode, it is returned as-is.

`robot.utils.encoding.console_encode(string, encoding=None, errors='replace',  
stream=<_io.TextIOWrapper name='<stdout>'>  
mode='w' encoding='UTF-8'>, force=False)`

Encodes the given string so that it can be used in the console.

If encoding is not given, determines it based on the given stream and system configuration. 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 decodes bytes back to Unicode because Python 3 APIs in general work with strings. Use *force=True* if that is not desired.

`robot.utils.encoding.system_decode(string)`

`robot.utils.encoding.system_encode(string)`

### 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(full_traceback=True, exclude_robot_traces=True)
    Returns error message and details of the last occurred exception.

class robot.utils.error.ErrorDetails(error=None, full_traceback=True, ex-
    clude_robot_traces=True)
    Bases: object

    Object wrapping the last occurred exception.

    It has attributes message, traceback, and error, where message contains the message with possible generic
    exception name removed, traceback contains the traceback and error contains the original error instance.

    message
    traceback
```

### robot.utils.escaping module

```
robot.utils.escaping.escape(item)
robot.utils.escaping.glob_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

```
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**

`robot.utils.frange.frange(*args)`

Like `range()` but accepts float arguments.

### **robot.utils.htmlformatters module**

`class robot.utils.htmlformatters.LinkFormatter`

Bases: object

`format_url(text)`

`format_link(text)`

`class robot.utils.htmlformatters.LineFormatter`

Bases: object

`handles(line)`

`newline = '\n'`

`format(line)`

`class robot.utils.htmlformatters.HtmlFormatter`

Bases: object

`format(text)`

`class robot.utils.htmlformatters.RulerFormatter`

Bases: `robot.utils.htmlformatters._SingleLineFormatter`

`match()`

Matches zero or more characters at the beginning of the string.

`format_line(line)`

`add(line)`

`end()`

`format(lines)`

`handles(line)`

`class robot.utils.htmlformatters.HeaderFormatter`

Bases: `robot.utils.htmlformatters._SingleLineFormatter`

```
match ()
    Matches zero or more characters at the beginning of the string.

format_line (line)

add (line)

end ()

format (lines)

handles (line)

class robot.utils.htmlformatters.ParagraphFormatter (other_formatters)
    Bases: robot.utils.htmlformatters._Formatter

    format (lines)

    add (line)

    end ()

    handles (line)

class robot.utils.htmlformatters.TableFormatter
    Bases: robot.utils.htmlformatters._Formatter

    format (lines)

    add (line)

    end ()

    handles (line)

class robot.utils.htmlformatters.PreformattedFormatter
    Bases: robot.utils.htmlformatters._Formatter

    format (lines)

    add (line)

    end ()

    handles (line)

class robot.utils.htmlformatters.ListFormatter
    Bases: robot.utils.htmlformatters._Formatter

    format (lines)

    add (line)

    end ()

    handles (line)
```

### robot.utils.importer module

```
class robot.utils.importer.Importer (type=None, logger=None)
    Bases: object

    Utility that can import modules and classes based on names and paths.

    Imported classes can optionally be instantiated automatically.
```

#### Parameters

- **type** – Type of the thing being imported. Used in error and log messages.
- **logger** – Logger to be notified about successful imports and other events. Currently only needs the `info` method, but other level specific methods may be needed in the future. If not given, logging is disabled.

**import\_class\_or\_module** (*name\_or\_path*, *instantiate\_with\_args=None*, *return\_source=False*)

Imports Python class or module based on the given name or path.

#### Parameters

- **name\_or\_path** – Name or path of the module or class to import.
- **instantiate\_with\_args** – When arguments are given, imported classes are automatically initialized using them.
- **return\_source** – When true, returns a tuple containing the imported module or class and a path to it. By default, returns only the imported module or class.

The class or module to import can be specified either as a name, in which case it must be in the module search path, or as a path to the file or directory implementing the module. See [`import\_class\_or\_module\_by\_path\(\)`](#) for more information about importing classes and modules by path.

Classes can be imported from the module search path using name like `modulename.ClassName`. If the class name and module name are same, using just `CommonName` is enough. When importing a class by a path, the class name and the module name must match.

Optional arguments to use when creating an instance are given as a list. Starting from Robot Framework 4.0, both positional and named arguments are supported (e.g. `['positional', 'name=value']`) and arguments are converted automatically based on type hints and default values.

If arguments needed when creating an instance are initially embedded into the name or path like Example:arg1:arg2, separate `split_args_from_name_or_path()` function can be used to split them before calling this method.

**import\_class\_or\_module\_by\_path** (*path*, *instantiate\_with\_args=None*)

Import a Python module or class using a file system path.

#### Parameters

- **path** – Path to the module or class to import.
- **instantiate\_with\_args** – When arguments are given, imported classes are automatically initialized using them.

When importing a Python file, the path must end with `.py` and the actual file must also exist.

Use [`import\_class\_or\_module\(\)`](#) to support importing also using name, not only path. See the documentation of that function for more information about creating instances 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)

class robot.utils.importer.NoLogger
    Bases: object

    error (*args, **kws)

    warn (*args, **kws)

    info (*args, **kws)

    debug (*args, **kws)

    trace (*args, **kws)
```

### 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)

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.

```
element (name, content=None, attrs=None, escape=True, newline=True)
```



```

close ()
    Closes the underlying output file.

content (content=None, escape=True, newline=False)

end (name, newline=True)

start (name, attrs=None, newline=True)

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

    Null implementation of the _MarkupWriter interface.

    start (**kwargs)

    content (**kwargs)

    element (**kwargs)

    end (**kwargs)

    close (**kwargs)

```

### robot.utils.match module

```

robot.utils.match.eq (str1, str2, ignore=(), caseless=True, spaceless=True)

class robot.utils.match.Matcher (pattern, ignore=(), caseless=True, spaceless=True, reg-
                                exp=False)
    Bases: object

    match (string)

    match_any (strings)

class robot.utils.match.MultiMatcher (patterns=None, ignore=(), caseless=True, space-
                                less=True, match_if_no_patterns=False, reg-
                                exp=False)
    Bases: object

    match (string)

    match_any (strings)

```

### robot.utils.misc module

```

robot.utils.misc.printable_name (string, code_style=False)
    Generates and returns printable name from the given string.

    Examples: 'simple' -> 'Simple' 'name with spaces' -> 'Name With Spaces' 'more spaces' -> 'More Spaces'
    'Cases AND spaces' -> 'Cases AND Spaces' " " -> "

    If 'code_style' is True:

    'mixedCAPSCamel' -> 'Mixed CAPS Camel' 'camelCaseName' -> 'Camel Case Name' 'under_score_name'
    -> 'Under Score Name' 'under_and space' -> 'Under And Space' 'miXed_CAPS_nAMe' -> 'MiXed CAPS
    NAME' " " -> "

robot.utils.misc.plural_or_not (item)

robot.utils.misc.seq2str (sequence, quote="", sep=', ', lastsep=' and ')
    Returns sequence in format 'item 1', 'item 2' and 'item 3'.

```

```
robot.utils.misc.seq2str2(sequence)
    Returns sequence in format [ item 1 | item 2 | ... ].

robot.utils.misc.test_or_task(text, rpa=False)
    Replaces {test} in text with test or task depending on rpa.

robot.utils.misc.isatty(stream)
```

### 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)

class robot.utils.normalizing.NormalizedDict(initial=None, ignore=(), caseless=True,
                                             spaceless=True)
    Bases: collections.abc.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() → a set-like object providing a view on D's items
keys() → a set-like object providing a view on 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() → an object providing a view on D's values
```

### robot.utils.platform module

```
robot.utils.platform.isatty(stream)
```

### robot.utils.recommendations module

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

    find_and_format (name, candidates, message, max_matches=10,
                    check_missing_argument_separator=False)

    find (name, candidates, max_matches=10)
        Return a list of close matches to name from candidates.

    format (message, recommendations)
        Add recommendations to the given message.

        The recommendation string looks like:
```

```
<message> Did you mean:
    <recommendations[0]>
    <recommendations[1]>
    <recommendations[2]>
```

### robot.utils.restreader module

### robot.utils.robotenv module

### robot.utils.robotinspect module

### robot.utils.robotio module

### robot.utils.robotpath module

### robot.utils.robottime module

### robot.utils.robottypes module

### robot.utils.setter module

### robot.utils.sortable module

### robot.utils.text module

### robot.utils.unic module

### robot.variables package

Implements storing and resolving variables.

This package is mainly for internal usage, but utilities for finding variables can be used externally as well.

## 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, mod-
                                                ules=None, namespace=None)

class robot.variables.evaluation.EvaluationNamespace (variable_store, namespace)
    Bases: collections.abc.Mapping

    get (k[, d]) → D[k] if k in D, else d. d defaults to None.

    items () → a set-like object providing a view on D's items

    keys () → a set-like object providing a view on D's keys

    values () → an object providing a view on D's values
```

**robot.variables.filesetter module**

```
class robot.variables.filesetter.VariableFileSetter(store)
    Bases: object

    set (path_or_variables, args=None, overwrite=False)

class robot.variables.filesetter.YamlImporter
    Bases: object

    import_variables (path, args=None)

class robot.variables.filesetter.PythonImporter
    Bases: object

    import_variables (path, args=None)
```

**robot.variables.finders module**

```
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 = '$@&'

    empty = <robot.utils.normalizing.NormalizedDict object>

    find (name)

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.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.

### robot.variables.replacer module

**class** robot.variables.replacer.**VariableReplacer**(*variable\_store*)  
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.resolvable module

**class** robot.variables.resolvable.**Resolvable**  
Bases: object

**resolve**(*variables*)

**report\_error**(*error*)

**class** robot.variables.resolvable.**GlobalVariableValue**(*value*)  
Bases: *robot.variables.resolvable.Resolvable*

**resolve**(*variables*)

**report\_error**(*error*)

### 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()
    end_suite()
    start_test()
    end_test()

```

```
start_keyword()  
end_keyword()  
set_global(name, value)  
set_suite(name, value)  
set_test(name, value)  
set_keyword(name, value)  
update(variables)
```

### robot.variables.search module

```
robot.variables.search.search_variable(string, identifiers='$@&%', ignore_errors=False)  
robot.variables.search.contains_variable(string, identifiers='$@&')  
robot.variables.search.is_variable(string, identifiers='$@&')  
robot.variables.search.is_scalar_variable(string)  
robot.variables.search.is_list_variable(string)  
robot.variables.search.is_dict_variable(string)  
robot.variables.search.is_assign(string, identifiers='$@&', allow_assign_mark=False)  
robot.variables.search.is_scalar_assign(string, allow_assign_mark=False)  
robot.variables.search.is_list_assign(string, allow_assign_mark=False)  
robot.variables.search.is_dict_assign(string, allow_assign_mark=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_scalar_variable()  
    is_list_variable()  
    is_dict_variable()  
    is_assign(allow_assign_mark=False)  
    is_scalar_assign(allow_assign_mark=False)  
    is_list_assign(allow_assign_mark=False)  
    is_dict_assign(allow_assign_mark=False)  
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='$@&%',      ig-
                                           nore_errors=False)
    Bases: object

```

### robot.variables.store module

```

class robot.variables.store.VariableStore(variables)
    Bases: object
    resolve_delayed(item=None)
    get(name, default=<object object>, decorated=True)
    update(store)
    clear()
    add(name, value, overwrite=True, decorated=True)
    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,      er-
                                                         ror_reporter=None)
    Bases: robot.variables.resolvable.Resolvable
    resolve(variables)
    report_error(error)
class robot.variables.tablesetter.ScalarVariableTableValue(values,      er-
                                                         ror_reporter=None)
    Bases: robot.variables.tablesetter.VariableTableValueBase
    report_error(error)
    resolve(variables)
class robot.variables.tablesetter.ListVariableTableValue(values,      er-
                                                         ror_reporter=None)
    Bases: robot.variables.tablesetter.VariableTableValueBase
    report_error(error)
    resolve(variables)

```

```
class robot.variables.tablesetter.DictVariableTableValue (values, er-  
                                                         ror_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)
```

### 3.1.2 Submodules

#### 3.1.3 robot.errors module

Exceptions and return codes used internally.

External libraries should not used exceptions defined here.

```
exception robot.errors.RobotError (message="", details="")  
    Bases: Exception  
    Base class for Robot Framework errors.  
    Do not raise this method but use more specific errors instead.  
    message  
    args  
    with_traceback ()  
        Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.
```

**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**

**with\_traceback()**

Exception.with\_traceback(tb) – set self.\_\_traceback\_\_ to tb and return self.

**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**

**with\_traceback()**

Exception.with\_traceback(tb) – set self.\_\_traceback\_\_ to tb and return self.

**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**

**with\_traceback()**

Exception.with\_traceback(tb) – set self.\_\_traceback\_\_ to tb and return self.

**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**

**with\_traceback()**

Exception.with\_traceback(tb) – set self.\_\_traceback\_\_ to tb and return self.

**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**

**with\_traceback()**

Exception.with\_traceback(tb) – set self.\_\_traceback\_\_ to tb and return self.

**exception** `robot.errors.Information` (*message="", details=""*)

Bases: `robot.errors.RobotError`

Used by argument parser with `-help` or `-version`.

**args**

**message**

**with\_traceback()**

Exception.with\_traceback(tb) – set self.\_\_traceback\_\_ to tb and return self.

**exception** `robot.errors.ExecutionStatus` (*message, test\_timeout=False, keyword\_timeout=False, syntax=False, exit=False, continue\_on\_failure=False, skip=False, return\_value=None*)

Bases: `robot.errors.RobotError`

Base class for exceptions communicating status in test execution.

**timeout**

**dont\_continue**

**continue\_on\_failure**

**can\_continue** (*context, templated=False*)

**get\_errors()**

**status**

**args**

**message**

**with\_traceback()**

Exception.with\_traceback(tb) – set self.\_\_traceback\_\_ to tb and return self.

**exception** `robot.errors.ExecutionFailed` (*message, test\_timeout=False, keyword\_timeout=False, syntax=False, exit=False, continue\_on\_failure=False, skip=False, return\_value=None*)

Bases: `robot.errors.ExecutionStatus`

Used for communicating failures in test execution.

**args**

**can\_continue** (*context, templated=False*)

**continue\_on\_failure**

**dont\_continue**

**get\_errors()**

**message**

**status**

**timeout**

```

with_traceback()
    Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception robot.errors.HandlerExecutionFailed(details)
    Bases: robot.errors.ExecutionFailed

    args
    can_continue(context, templated=False)
    continue_on_failure
    dont_continue
    get_errors()
    message
    status
    timeout
    with_traceback()
        Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception robot.errors.ExecutionFailures(errors, message=None)
    Bases: robot.errors.ExecutionFailed

    get_errors()
    args
    can_continue(context, templated=False)
    continue_on_failure
    dont_continue
    message
    status
    timeout
    with_traceback()
        Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception robot.errors.UserKeywordExecutionFailed(run_errors=None, down_errors=None) tear-
    Bases: robot.errors.ExecutionFailures

    args
    can_continue(context, templated=False)
    continue_on_failure
    dont_continue
    get_errors()
    message
    status
    timeout
    with_traceback()
        Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

```

**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** (*context, templated=False*)

**continue\_on\_failure**

**dont\_continue**

**get\_errors** ()

**message**

**timeout**

**with\_traceback** ()

Exception.with\_traceback(tb) – set self.\_\_traceback\_\_ to tb and return self.

**exception** `robot.errors.PassExecution` (*message*)

Bases: `robot.errors.ExecutionPassed`

Used by ‘Pass Execution’ keyword.

**args**

**can\_continue** (*context, templated=False*)

**continue\_on\_failure**

**dont\_continue**

**earlier\_failures**

**get\_errors** ()

**message**

**set\_earlier\_failures** (*failures*)

**status**

**timeout**

**with\_traceback** ()

Exception.with\_traceback(tb) – set self.\_\_traceback\_\_ to tb and return self.

**exception** `robot.errors.ContinueLoop`

Bases: `robot.errors.ExecutionPassed`

Used by CONTINUE statement.

**args**

**can\_continue** (*context, templated=False*)

**continue\_on\_failure**

**dont\_continue**

```

    earlier_failures
    get_errors ()
    message
    set_earlier_failures (failures)
    status
    timeout
    with_traceback ()
        Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.
exception robot.errors.BreakLoop
    Bases: robot.errors.ExecutionPassed
    Used by BREAK statement.
    args
    can_continue (context, templated=False)
    continue_on_failure
    dont_continue
    earlier_failures
    get_errors ()
    message
    set_earlier_failures (failures)
    status
    timeout
    with_traceback ()
        Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.
exception robot.errors.ReturnFromKeyword (return_value=None, failures=None)
    Bases: robot.errors.ExecutionPassed
    Used by 'RETURN' statement.
    args
    can_continue (context, templated=False)
    continue_on_failure
    dont_continue
    earlier_failures
    get_errors ()
    message
    set_earlier_failures (failures)
    status
    timeout
    with_traceback ()
        Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

```

**exception** `robot.errors.RemoteError` (*message=""*, *details=""*, *fatal=False*, *continuable=False*)

Bases: `robot.errors.RobotError`

Used by Remote library to report remote errors.

**args**

**message**

**with\_traceback()**

Exception.with\_traceback(tb) – set self.\_\_traceback\_\_ to tb and return self.

### 3.1.4 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*, *specdocformat=None*, *python-path=None*, *quiet=False*)

**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=None*, *exit=True*)

Executes Libdoc similarly as from the command line.

**Parameters**

- **arguments** – Command line options and arguments as a list of strings. Starting from RF 4.0, defaults to `sys.argv[1:]` if not given.
- **exit** – If True, call `sys.exit` automatically. New in RF 4.0.

The `libdoc()` function may work better in programmatic usage.

Example:



```
from robot.libdoc import libdoc_cli

libdoc_cli(['--version', '1.0', 'MyLibrary.py', 'MyLibrary.html'])
```

`robot.libdoc.libdoc` (*library\_or\_resource*, *outfile*, *name*=", *version*", *format*=None, *docformat*=None, *specdocformat*=None, *quiet*=False)

Executes Libdoc.

#### Parameters

- **library\_or\_resource** – Name or path of the 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, XML or JSON output. If this options is not used, the format is got from the extension of the output file. Possible values are 'HTML', 'XML', 'JSON' and 'LIBSPEC'.
- **docformat** – Documentation source format. Possible values are 'ROBOT', 'reST', 'HTML' and 'TEXT'. The default value can be specified in library source code and the initial default is 'ROBOT'.
- **specdocformat** – Specifies whether the keyword documentation in spec files is converted to HTML regardless of the original documentation format. Possible values are 'HTML' (convert to HTML) and 'RAW' (use original format). The default depends on the output format. New in Robot Framework 4.0.
- **quiet** – When true, the path of the generated output file is not printed the console. New in Robot Framework 4.0.

Arguments have same semantics as Libdoc command line options with same names. Run `libdoc --help` or consult the Libdoc section in the Robot Framework User Guide for more details.

Example:

```
from robot.libdoc import libdoc

libdoc('MyLibrary.py', 'MyLibrary.html', version='1.0')
```

### 3.1.5 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*)

### 3.1.6 robot.rebot module

Module implementing the command line entry point for post-processing outputs.

This module can be executed from the command line using the following approaches:

```
python -m robot.rebot
python path/to/robot/rebot.py
```

Instead of `python` it is possible to use also other Python interpreters. This module is also used by the installed `rebot` start-up script.

This module also provides `rebot()` and `rebot_cli()` functions that can be used programmatically. Other code is for internal usage.

**class** `robot.rebot.Rebot`

Bases: `robot.run.RobotFramework`

**main** (`datasources`, `**options`)

**console** (`msg`)

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

**execute\_cli** (`cli_arguments`, `exit=True`)

**parse\_arguments** (`cli_args`)

Public interface for parsing command line arguments.

**Parameters** `cli_args` – Command line arguments as a list

**Returns** options (dict), arguments (list)

**Raises** `Information` when `--help` or `--version` used

**Raises** `DataError` when parsing fails

**validate** (`options`, `arguments`)

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

Command line execution entry point for post-processing outputs.

**Parameters**

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

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

Example:

```
from robot import rebot_cli

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

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

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

Programmatic entry point for post-processing outputs.

**Parameters**

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

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

Examples:

```
from robot import rebot

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

Equivalent command line usage:

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

### 3.1.7 robot.run module

Module implementing the command line entry point for executing tests.

This module can be executed from the command line using the following approaches:

```
python -m robot.run
python path/to/robot/run.py
```

Instead of `python` it is possible to use also other Python interpreters. This module is also used by the installed `robot` start-up script.

This module also provides `run()` and `run_cli()` functions that can be used programmatically. Other code is for internal usage.

**class** `robot.run.RobotFramework`

Bases: `robot.utils.application.Application`

**main** (*datasources*, *\*\*options*)

**validate** (*options*, *arguments*)

**console** (*msg*)

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

**execute\_cli** (*cli\_arguments*, *exit=True*)

**parse\_arguments** (*cli\_args*)

Public interface for parsing command line arguments.

**Parameters** `cli_args` – Command line arguments as a list

**Returns** options (dict), arguments (list)

**Raises** `Information` when `-help` or `-version` used

**Raises** `DataError` when parsing fails

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

Command line execution entry point for running tests.

**Parameters**

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

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

Example:

```
from robot import run_cli

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

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

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

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

Programmatic entry point for running tests.

#### Parameters

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

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

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

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

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

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

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

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

Example:

```
from robot import run

run('path/to/tests.robot')
```

(continues on next page)

(continued from previous page)

```
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
```

### 3.1.8 robot.testdoc module

Module implementing the command line entry point for the *Testdoc* tool.

This module can be executed from the command line using the following approaches:

```
python -m robot.testdoc
python path/to/robot/testdoc.py
```

Instead of python it is possible to use also other Python interpreters.

This module also provides `testdoc()` and `testdoc_cli()` functions that can be used programmatically. Other code is for internal usage.

**class** `robot.testdoc.TestDoc`

Bases: `robot.utils.application.Application`

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

**console** (*msg*)

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

**execute\_cli** (*cli\_arguments*, *exit=True*)

**parse\_arguments** (*cli\_args*)

Public interface for parsing command line arguments.

**Parameters** *cli\_args* – Command line arguments as a list

**Returns** options (dict), arguments (list)

**Raises** `Information` when `-help` or `-version` used

**Raises** `DataError` when parsing fails

**validate** (*options*, *arguments*)

`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')
```

### 3.1.9 robot.version module

`robot.version.get_version` (*naked=False*)

`robot.version.get_full_version` (*program=None, naked=False*)

`robot.version.get_interpreter` ()

## CHAPTER 4

---

### Indices

---

- `genindex`
- `modindex`
- `search`





### r

- `robot`, 7
- `robot.api`, 5
- `robot.api.deco`, 10
- `robot.api.exceptions`, 12
- `robot.api.logger`, 14
- `robot.api.parsing`, 15
- `robot.conf`, 22
- `robot.conf.gatherfailed`, 23
- `robot.conf.settings`, 31
- `robot.errors`, 574
- `robot.htmldata`, 33
- `robot.htmldata.htmlfilewriter`, 33
- `robot.htmldata.jsonwriter`, 34
- `robot.htmldata.template`, 35
- `robot.libdoc`, 580
- `robot.libdocpkg`, 35
- `robot.libdocpkg.builder`, 35
- `robot.libdocpkg.consoleviewer`, 36
- `robot.libdocpkg.datatypes`, 36
- `robot.libdocpkg.htmlutils`, 36
- `robot.libdocpkg.htmlwriter`, 37
- `robot.libdocpkg.jsonbuilder`, 37
- `robot.libdocpkg.jsonwriter`, 37
- `robot.libdocpkg.model`, 37
- `robot.libdocpkg.output`, 38
- `robot.libdocpkg.robotbuilder`, 38
- `robot.libdocpkg.standardtypes`, 38
- `robot.libdocpkg.writer`, 38
- `robot.libdocpkg.xmlbuilder`, 38
- `robot.libdocpkg.xmlwriter`, 39
- `robot.libraries`, 39
- `robot.libraries.BuiltIn`, 39
- `robot.libraries.Collections`, 65
- `robot.libraries.DateTime`, 72
- `robot.libraries.Dialogs`, 77
- `robot.libraries.dialogs_py`, 123
- `robot.libraries.Easter`, 77
- `robot.libraries.OperatingSystem`, 78
- `robot.libraries.Process`, 87
- `robot.libraries.Remote`, 94
- `robot.libraries.Reserved`, 95
- `robot.libraries.Screenshot`, 95
- `robot.libraries.String`, 97
- `robot.libraries.Telnet`, 103
- `robot.libraries.XML`, 112
- `robot.model`, 193
- `robot.model.body`, 193
- `robot.model.configurer`, 199
- `robot.model.control`, 203
- `robot.model.filter`, 215
- `robot.model.fixture`, 223
- `robot.model.itemlist`, 224
- `robot.model.keyword`, 224
- `robot.model.message`, 227
- `robot.model.metadata`, 229
- `robot.model.modelobject`, 230
- `robot.model.modifier`, 230
- `robot.model.namepatterns`, 235
- `robot.model.statistics`, 235
- `robot.model.stats`, 239
- `robot.model.suitestatistics`, 241
- `robot.model.tags`, 242
- `robot.model.tagsetter`, 242
- `robot.model.tagstatistics`, 247
- `robot.model.testcase`, 247
- `robot.model.testsuite`, 250
- `robot.model.totalstatistics`, 253
- `robot.model.visitor`, 258
- `robot.output`, 263
- `robot.output.console`, 263
- `robot.output.console.dotted`, 263
- `robot.output.console.highlighting`, 268
- `robot.output.console.quiet`, 269
- `robot.output.console.verbose`, 269
- `robot.output.debugfile`, 270
- `robot.output.filelogger`, 270
- `robot.output.librarylogger`, 270
- `robot.output.listenerarguments`, 271

`robot.output.listenermethods`, 272  
`robot.output.listeners`, 272  
`robot.output.logger`, 273  
`robot.output.loggerhelper`, 274  
`robot.output.output`, 276  
`robot.output.pyloggingconf`, 277  
`robot.output.stdoutlogsplitter`, 278  
`robot.output.xmllogger`, 278  
`robot.parsing`, 283  
`robot.parsing.lexer`, 283  
`robot.parsing.lexer.blocklexers`, 283  
`robot.parsing.lexer.context`, 288  
`robot.parsing.lexer.lexer`, 290  
`robot.parsing.lexer.sections`, 290  
`robot.parsing.lexer.settings`, 292  
`robot.parsing.lexer.statementlexers`, 293  
`robot.parsing.lexer.tokenizer`, 299  
`robot.parsing.lexer.tokens`, 299  
`robot.parsing.model`, 306  
`robot.parsing.model.blocks`, 306  
`robot.parsing.model.statements`, 311  
`robot.parsing.model.visitor`, 353  
`robot.parsing.parser`, 353  
`robot.parsing.parser.blockparsers`, 353  
`robot.parsing.parser.fileparser`, 354  
`robot.parsing.parser.parser`, 356  
`robot.parsing.suitestructure`, 356  
`robot.pythonpathsetter`, 581  
`robot.rebot`, 581  
`robot.reporting`, 357  
`robot.reporting.expandkeywordmatcher`, 357  
`robot.reporting.jsbuildingcontext`, 357  
`robot.reporting.jsexecutionresult`, 358  
`robot.reporting.jsmodelbuilders`, 358  
`robot.reporting.jswriter`, 359  
`robot.reporting.logreportwriters`, 359  
`robot.reporting.outputwriter`, 359  
`robot.reporting.resultwriter`, 364  
`robot.reporting.stringcache`, 365  
`robot.reporting.xunitwriter`, 366  
`robot.result`, 371  
`robot.result.configurer`, 372  
`robot.result.executionerrors`, 376  
`robot.result.executionresult`, 377  
`robot.result.flattenkeywordmatcher`, 379  
`robot.result.keywordremover`, 379  
`robot.result.merger`, 413  
`robot.result.messagefilter`, 417  
`robot.result.model`, 421  
`robot.result.modeldeprecation`, 463  
`robot.result.resultbuilder`, 464  
`robot.result.suiteteardownfailed`, 469  
`robot.result.visitor`, 477  
`robot.result.xmlelementhandlers`, 483  
`robot.run`, 583  
`robot.running`, 491  
`robot.running.arguments`, 492  
`robot.running.arguments.argumentconverter`, 492  
`robot.running.arguments.argumentmapper`, 492  
`robot.running.arguments.argumentparser`, 493  
`robot.running.arguments.argumentresolver`, 493  
`robot.running.arguments.argumentspec`, 494  
`robot.running.arguments.argumentvalidator`, 494  
`robot.running.arguments.customconverters`, 495  
`robot.running.arguments.embedded`, 495  
`robot.running.arguments.typeconverters`, 495  
`robot.running.arguments.typevalidator`, 503  
`robot.running.bodyrunner`, 511  
`robot.running.builder`, 503  
`robot.running.builder.builders`, 504  
`robot.running.builder.parsers`, 505  
`robot.running.builder.testsettings`, 505  
`robot.running.builder.transformers`, 506  
`robot.running.context`, 513  
`robot.running.dynamicmethods`, 513  
`robot.running.handlers`, 513  
`robot.running.handlerstore`, 514  
`robot.running.importer`, 514  
`robot.running.librarykeywordrunner`, 514  
`robot.running.libraryscopes`, 515  
`robot.running.model`, 516  
`robot.running.modelcombiner`, 539  
`robot.running.namespace`, 540  
`robot.running.outputcapture`, 540  
`robot.running.randomizer`, 541  
`robot.running.runkwregister`, 545  
`robot.running.signalhandler`, 545  
`robot.running.status`, 545  
`robot.running.statusreporter`, 547  
`robot.running.suiterunner`, 547  
`robot.running.testlibraries`, 551  
`robot.running.timeouts`, 510  
`robot.running.timeouts.posix`, 511  
`robot.running.timeouts.windows`, 511  
`robot.running.usererrorhandler`, 551  
`robot.running.userkeyword`, 552  
`robot.running.userkeywordrunner`, 552  
`robot.testdoc`, 585

- robot.utils, [553](#)
- robot.utils.application, [553](#)
- robot.utils.argumentparser, [554](#)
- robot.utils.asserts, [555](#)
- robot.utils.charwidth, [557](#)
- robot.utils.compress, [557](#)
- robot.utils.connectioncache, [557](#)
- robot.utils.dotdict, [558](#)
- robot.utils.encoding, [559](#)
- robot.utils.encodingsniffer, [560](#)
- robot.utils.error, [560](#)
- robot.utils.escaping, [560](#)
- robot.utils.etreewrapper, [560](#)
- robot.utils.filereader, [560](#)
- robot.utils.frange, [561](#)
- robot.utils.htmlformatters, [561](#)
- robot.utils.importer, [562](#)
- robot.utils.markuputils, [564](#)
- robot.utils.markupwriters, [564](#)
- robot.utils.match, [565](#)
- robot.utils.misc, [565](#)
- robot.utils.normalizing, [566](#)
- robot.utils.platform, [566](#)
- robot.utils.recommendations, [567](#)
- robot.variables, [567](#)
- robot.variables.assigner, [568](#)
- robot.variables.evaluation, [568](#)
- robot.variables.filesetter, [569](#)
- robot.variables.finders, [569](#)
- robot.variables.notfound, [570](#)
- robot.variables.replacer, [570](#)
- robot.variables.resolvable, [570](#)
- robot.variables.scopes, [570](#)
- robot.variables.search, [572](#)
- robot.variables.store, [573](#)
- robot.variables.tablesetter, [573](#)
- robot.variables.variables, [574](#)
- robot.version, [586](#)



## A

- abc (robot.running.arguments.typeconverters.BooleanConverter attribute), 496
- abc (robot.running.arguments.typeconverters.ByteArrayConverter attribute), 498
- abc (robot.running.arguments.typeconverters.BytesConverter attribute), 498
- abc (robot.running.arguments.typeconverters.CombinedConverter attribute), 502
- abc (robot.running.arguments.typeconverters.CustomConverter attribute), 503
- abc (robot.running.arguments.typeconverters.DateConverter attribute), 499
- abc (robot.running.arguments.typeconverters.DateTimeConverter attribute), 499
- abc (robot.running.arguments.typeconverters.DecimalConverter attribute), 498
- abc (robot.running.arguments.typeconverters.DictionaryConverter attribute), 501
- abc (robot.running.arguments.typeconverters.EnumConverter attribute), 496
- abc (robot.running.arguments.typeconverters.FloatConverter attribute), 497
- abc (robot.running.arguments.typeconverters.FrozenSetConverter attribute), 502
- abc (robot.running.arguments.typeconverters.IntegerConverter attribute), 497
- abc (robot.running.arguments.typeconverters.ListConverter attribute), 500
- abc (robot.running.arguments.typeconverters.NoneConverter attribute), 500
- abc (robot.running.arguments.typeconverters.SetConverter attribute), 502
- abc (robot.running.arguments.typeconverters.StringConverter attribute), 496
- abc (robot.running.arguments.typeconverters.TimeDeltaConverter attribute), 500
- abc (robot.running.arguments.typeconverters.TupleConverter attribute), 501
- abc (robot.running.arguments.typeconverters.TypeConverter attribute), 495
- AbstractLogger (class in robot.output.loggerhelper), 274
- AbstractLoggerProxy (class in robot.output.loggerhelper), 276
- accept\_gzip\_encoding (robot.libraries.Remote.TimeoutHTTPSTransport attribute), 95
- accept\_gzip\_encoding (robot.libraries.Remote.TimeoutHTTPSTransport attribute), 94
- accepts\_more() (robot.parsing.lexer.blocklexers.BlockLexer method), 283
- accepts\_more() (robot.parsing.lexer.blocklexers.CommentSectionLexer method), 285
- accepts\_more() (robot.parsing.lexer.blocklexers.ErrorSectionLexer method), 286
- accepts\_more() (robot.parsing.lexer.blocklexers.FileLexer method), 284
- accepts\_more() (robot.parsing.lexer.blocklexers.ForLexer method), 287
- accepts\_more() (robot.parsing.lexer.blocklexers.IfLexer method), 287
- accepts\_more() (robot.parsing.lexer.blocklexers.ImplicitCommentSectionLexer method), 285
- accepts\_more() (robot.parsing.lexer.blocklexers.InlineIfLexer method), 287
- accepts\_more() (robot.parsing.lexer.blocklexers.KeywordLexer method), 286
- accepts\_more() (robot.parsing.lexer.blocklexers.KeywordSectionLexer method), 285
- accepts\_more() (robot.parsing.lexer.blocklexers.NestedBlockLexer method), 286
- accepts\_more() (robot.parsing.lexer.blocklexers.SectionLexer method), 284
- accepts\_more() (robot.parsing.lexer.blocklexers.SettingSectionLexer method), 284
- accepts\_more() (robot.parsing.lexer.blocklexers.TestCaseLexer method), 286

`accepts_more()` (`(robot.parsing.lexer.blocklexers.TestCaseSectionHeaderLexer method)`, 285  
`accepts_more()` (`(robot.parsing.lexer.blocklexers.TestOrKeywordLexer method)`, 286  
`accepts_more()` (`(robot.parsing.lexer.blocklexers.TryLexer method)`, 288  
`accepts_more()` (`(robot.parsing.lexer.blocklexers.VariableSectionHeaderLexer method)`, 284  
`accepts_more()` (`(robot.parsing.lexer.blocklexers.WhileLexer method)`, 287  
`accepts_more()` (`(robot.parsing.lexer.statementlexers.BracketedText method)`, 299  
`accepts_more()` (`(robot.parsing.lexer.statementlexers.CommentLexer method)`, 295  
`accepts_more()` (`(robot.parsing.lexer.statementlexers.CommentSectionHeaderLexer method)`, 295  
`accepts_more()` (`(robot.parsing.lexer.statementlexers.ContinuedText method)`, 298  
`accepts_more()` (`(robot.parsing.lexer.statementlexers.ElseHeaderLexer method)`, 297  
`accepts_more()` (`(robot.parsing.lexer.statementlexers.ElseIfHeaderLexer method)`, 297  
`accepts_more()` (`(robot.parsing.lexer.statementlexers.EndLexer method)`, 298  
`accepts_more()` (`(robot.parsing.lexer.statementlexers.ErrorSectionHeaderLexer method)`, 295  
`accepts_more()` (`(robot.parsing.lexer.statementlexers.ExceptHeaderLexer method)`, 298  
`accepts_more()` (`(robot.parsing.lexer.statementlexers.FinallyHeaderLexer method)`, 298  
`accepts_more()` (`(robot.parsing.lexer.statementlexers.ForHeaderLexer method)`, 296  
`accepts_more()` (`(robot.parsing.lexer.statementlexers.IfHeaderLexer method)`, 297  
`accepts_more()` (`(robot.parsing.lexer.statementlexers.InlineIfHeaderLexer method)`, 297  
`accepts_more()` (`(robot.parsing.lexer.statementlexers.KeywordCallMethod method)`, 296  
`accepts_more()` (`(robot.parsing.lexer.statementlexers.KeywordSectionHeaderLexer method)`, 295  
`accepts_more()` (`(robot.parsing.lexer.statementlexers.Lexer method)`, 293  
`accepts_more()` (`(robot.parsing.lexer.statementlexers.ReadText method)`, 298  
`accepts_more()` (`(robot.parsing.lexer.statementlexers.SectionHeaderLexer method)`, 294  
`accepts_more()` (`(robot.parsing.lexer.statementlexers.SettingLexer method)`, 296  
`accepts_more()` (`(robot.parsing.lexer.statementlexers.SettingSectionHeaderLexer method)`, 294  
`accepts_more()` (`(robot.parsing.lexer.statementlexers.SingleTags method)`, 294  
`accepts_more()` (`(robot.parsing.lexer.statementlexers.StatementText method)`, 294

`add()` (`(robot.reporting.stringcache.StringCache method)`, 366  
`add()` (`(robot.result.executionerrors.ExecutionErrors method)`, 377  
`add()` (`(robot.running.handlerstore.HandlerStore method)`, 514  
`add()` (`(robot.running.importer.ImportCache method)`, 514  
`add()` (`(robot.utils.htmlformatters.HeaderFormatter method)`, 562  
`add()` (`(robot.utils.htmlformatters.ListFormatter method)`, 562  
`add()` (`(robot.utils.htmlformatters.ParagraphFormatter method)`, 562  
`add()` (`(robot.utils.htmlformatters.PreformattedFormatter method)`, 562  
`add()` (`(robot.utils.htmlformatters.RulerFormatter method)`, 562  
`add()` (`(robot.utils.htmlformatters.TableFormatter method)`, 562  
`add()` (`(robot.variables.store.VariableStore method)`, 573  
`add_result()` (`(robot.libraries.XML.XML method)`, 120  
`from module robot.pythonpathsetter`, 581  
`get_result()` (`(robot.result.executionresult.CombinedResult method)`, 378  
`stat_stat()` (`(robot.model.stats.SuiteStat method)`, 240  
`suite_configurer` (`(robot.model.configurer.SuiteConfigurer attribute)`, 199  
`suite_configurer` (`(robot.result.configurer.SuiteConfigurer attribute)`, 372  
`tag_stat` (`(robot.model.stats.CombinedTagStat method)`, 241

- `add_test()` (`robot.model.stats.Stat` method), 240  
`add_test()` (`robot.model.stats.SuiteStat` method), 240  
`add_test()` (`robot.model.stats.TagStat` method), 241  
`add_test()` (`robot.model.stats.TotalStat` method), 240  
`add_test()` (`robot.model.sitestatistics.SuiteStatisticsBuilder` method), 242  
`add_test()` (`robot.model.tagstatistics.TagStatisticsBuilder` method), 247  
`add_test()` (`robot.model.totalstatistics.TotalStatisticsBuilder` method), 253  
`add_test()` (`robot.model.totalstatistics.TotalStatisticsBuilder` method), 253  
`add_time_to_date()` (in `robot.libraries.DateTime` module), 75  
`add_time_to_time()` (in `robot.libraries.DateTime` module), 76  
`addFilter()` (`robot.output.pyloggingconf.RobotHandler` method), 277  
`after()` (`robot.variables.search.VariableMatch` attribute), 572  
`after()` (`robot.libraries.dialogs_py.InputDialog` method), 137  
`after()` (`robot.libraries.dialogs_py.MessageDialog` method), 123  
`after()` (`robot.libraries.dialogs_py.MultipleSelectionDialog` method), 165  
`after()` (`robot.libraries.dialogs_py.PassFailDialog` method), 179  
`after()` (`robot.libraries.dialogs_py.SelectionDialog` method), 151  
`after_cancel()` (`robot.libraries.dialogs_py.InputDialog` method), 137  
`after_cancel()` (`robot.libraries.dialogs_py.MessageDialog` method), 123  
`after_cancel()` (`robot.libraries.dialogs_py.MultipleSelectionDialog` method), 165  
`after_cancel()` (`robot.libraries.dialogs_py.PassFailDialog` method), 179  
`after_cancel()` (`robot.libraries.dialogs_py.SelectionDialog` method), 151  
`after_idle()` (`robot.libraries.dialogs_py.InputDialog` method), 137  
`after_idle()` (`robot.libraries.dialogs_py.MessageDialog` method), 123  
`after_idle()` (`robot.libraries.dialogs_py.MultipleSelectionDialog` method), 165  
`after_idle()` (`robot.libraries.dialogs_py.PassFailDialog` method), 179  
`after_idle()` (`robot.libraries.dialogs_py.SelectionDialog` method), 151  
`alias` (`robot.parsing.model.statements.LibraryImport` attribute), 316  
`aliases` (`robot.parsing.lexer.settings.InitFileSettings` attribute), 292  
`aliases` (`robot.parsing.lexer.settings.KeywordSettings` attribute), 293  
`aliases` (`robot.parsing.lexer.settings.ResourceFileSettings` attribute), 293  
`aliases` (`robot.parsing.lexer.settings.Settings` attribute), 292  
`aliases` (`robot.parsing.lexer.settings.TestCaseFileSettings` attribute), 292  
`aliases` (`robot.parsing.lexer.settings.TestCaseSettings` attribute), 293  
`aliases` (`robot.running.arguments.typeconverters.BooleanConverter` attribute), 496  
`aliases` (`robot.running.arguments.typeconverters.ByteArrayConverter` attribute), 498  
`aliases` (`robot.running.arguments.typeconverters.BytesConverter` attribute), 498  
`aliases` (`robot.running.arguments.typeconverters.CombinedConverter` attribute), 502  
`aliases` (`robot.running.arguments.typeconverters.CustomConverter` attribute), 503  
`aliases` (`robot.running.arguments.typeconverters.DateConverter` attribute), 499  
`aliases` (`robot.running.arguments.typeconverters.DateTimeConverter` attribute), 499  
`aliases` (`robot.running.arguments.typeconverters.DecimalConverter` attribute), 498  
`aliases` (`robot.running.arguments.typeconverters.DictionaryConverter` attribute), 501  
`aliases` (`robot.running.arguments.typeconverters.EnumConverter` attribute), 496  
`aliases` (`robot.running.arguments.typeconverters.FloatConverter` attribute), 497  
`aliases` (`robot.running.arguments.typeconverters.FrozenSetConverter` attribute), 502  
`aliases` (`robot.running.arguments.typeconverters.IntegerConverter` attribute), 497  
`aliases` (`robot.running.arguments.typeconverters.ListConverter` attribute), 501  
`aliases` (`robot.running.arguments.typeconverters.NoneConverter` attribute), 500  
`aliases` (`robot.running.arguments.typeconverters.SetConverter` attribute), 502  
`aliases` (`robot.running.arguments.typeconverters.StringConverter` attribute), 496  
`aliases` (`robot.running.arguments.typeconverters.TimeDeltaConverter` attribute), 500  
`aliases` (`robot.running.arguments.typeconverters.TupleConverter` attribute), 501  
`aliases` (`robot.running.arguments.typeconverters.TypeConverter` attribute), 495  
`all` (`robot.model.keyword.Keywords` attribute), 226  
`all_tags` (`robot.libdocpkg.model.LibraryDoc` attribute), 38  
`AllKeywordsRemover` (class in



*robot.result.keywordremover*), 379

`ALLOW_VARIABLES` (*robot.parsing.lexer.tokens.END attribute*), 303

`ALLOW_VARIABLES` (*robot.parsing.lexer.tokens.EOS attribute*), 301

`ALLOW_VARIABLES` (*robot.parsing.lexer.tokens.Token attribute*), 301

`ALLOWED_TYPES` (*robot.running.model.Import attribute*), 539

`also_teardown_message` (*robot.running.status.ParentMessage attribute*), 546

`also_teardown_message` (*robot.running.status.SuiteMessage attribute*), 546

`also_teardown_message` (*robot.running.status.TestMessage attribute*), 546

`also_teardown_skip_message` (*robot.running.status.ParentMessage attribute*), 546

`also_teardown_skip_message` (*robot.running.status.SuiteMessage attribute*), 546

`also_teardown_skip_message` (*robot.running.status.TestMessage attribute*), 546

`anchor()` (*robot.libraries.dialogs\_py.InputDialog method*), 137

`anchor()` (*robot.libraries.dialogs\_py.MessageDialog method*), 123

`anchor()` (*robot.libraries.dialogs\_py.MultipleSelectionDialog method*), 165

`anchor()` (*robot.libraries.dialogs\_py.PassFailDialog method*), 179

`anchor()` (*robot.libraries.dialogs\_py.SelectionDialog method*), 151

`AndTagPattern` (class in *robot.model.tags*), 242

`AnsiHighlighter` (class in *robot.output.console.highlighting*), 268

`any_timeout_occurred()` (*robot.running.timeouts.TestTimeout method*), 510

`append()` (*robot.model.body.BaseBody method*), 195

`append()` (*robot.model.body.Body method*), 196

`append()` (*robot.model.body.Branches method*), 197

`append()` (*robot.model.itemlist.ItemList method*), 224

`append()` (*robot.model.keyword.Keywords method*), 226

`append()` (*robot.model.message.Messages method*), 229

`append()` (*robot.model.testcase.TestCases method*), 249

`append()` (*robot.model.testsuite.TestSuites method*), 252

`append()` (*robot.result.model.Body method*), 422

`append()` (*robot.result.model.Branches method*), 423

`append()` (*robot.result.model.Iterations method*), 425

`append()` (*robot.running.model.Body method*), 516

`append()` (*robot.running.model.Imports method*), 539

`append_to_environment_variable()` (*robot.libraries.OperatingSystem.OperatingSystem method*), 84

`append_to_file()` (*robot.libraries.OperatingSystem.OperatingSystem method*), 82

`append_to_list()` (*robot.libraries.Collections.Collections method*), 67

`Application` (class in *robot.utils.application*), 553

`ArgFileParser` (class in *robot.utils.argumentparser*), 554

`ArgInfo` (class in *robot.running.arguments.argumentspec*), 494

`ArgLimitValidator` (class in *robot.utils.argumentparser*), 554

`args` (*robot.api.exceptions.ContinuableFailure attribute*), 13

`args` (*robot.api.exceptions.Error attribute*), 13

`args` (*robot.api.exceptions.Failure attribute*), 12

`args` (*robot.api.exceptions.FatalError attribute*), 13

`args` (*robot.api.exceptions.SkipExecution attribute*), 14

`args` (*robot.errors.BreakLoop attribute*), 579

`args` (*robot.errors.ContinueLoop attribute*), 578

`args` (*robot.errors.DataError attribute*), 575

`args` (*robot.errors.ExecutionFailed attribute*), 576

`args` (*robot.errors.ExecutionFailures attribute*), 577

`args` (*robot.errors.ExecutionPassed attribute*), 578

`args` (*robot.errors.ExecutionStatus attribute*), 576

`args` (*robot.errors.FrameworkError attribute*), 575

`args` (*robot.errors.HandlerExecutionFailed attribute*), 577

`args` (*robot.errors.Information attribute*), 576

`args` (*robot.errors.KeywordError attribute*), 575

`args` (*robot.errors.PassExecution attribute*), 578

`args` (*robot.errors.RemoteError attribute*), 580

`args` (*robot.errors.ReturnFromKeyword attribute*), 579

`args` (*robot.errors.RobotError attribute*), 574

`args` (*robot.errors.TimeoutError attribute*), 575

`args` (*robot.errors.UserKeywordExecutionFailed attribute*), 577

`args` (*robot.errors.VariableError attribute*), 575

`args` (*robot.libraries.BuiltIn.RobotNotRunningError attribute*), 65

`args` (*robot.libraries.Telnet.NoMatchError attribute*), 112

`args` (*robot.model.keyword.Keyword attribute*), 224

`args` (*robot.parsing.model.statements.Fixture attribute*), 314



[args \(robot.parsing.model.statements.KeywordCall attribute\), 335](#)  
[args \(robot.parsing.model.statements.LibraryImport attribute\), 316](#)  
[args \(robot.parsing.model.statements.Setup attribute\), 329](#)  
[args \(robot.parsing.model.statements.SuiteSetup attribute\), 322](#)  
[args \(robot.parsing.model.statements.SuiteTeardown attribute\), 323](#)  
[args \(robot.parsing.model.statements.Teardown attribute\), 330](#)  
[args \(robot.parsing.model.statements.TemplateArguments attribute\), 336](#)  
[args \(robot.parsing.model.statements.TestSetup attribute\), 324](#)  
[args \(robot.parsing.model.statements.TestTeardown attribute\), 325](#)  
[args \(robot.parsing.model.statements.VariablesImport attribute\), 318](#)  
[args \(robot.result.model.Break attribute\), 452](#)  
[args \(robot.result.model.Continue attribute\), 449](#)  
[args \(robot.result.model.For attribute\), 432](#)  
[args \(robot.result.model.ForIteration attribute\), 430](#)  
[args \(robot.result.model.If attribute\), 441](#)  
[args \(robot.result.model.IfBranch attribute\), 439](#)  
[args \(robot.result.model.Keyword attribute\), 455](#)  
[args \(robot.result.model.Return attribute\), 447](#)  
[args \(robot.result.model.Try attribute\), 445](#)  
[args \(robot.result.model.TryBranch attribute\), 443](#)  
[args \(robot.result.model.While attribute\), 436](#)  
[args \(robot.result.model.WhileIteration attribute\), 434](#)  
[args \(robot.result.model.deprecation.DeprecatedAttributesMixin attribute\), 464](#)  
[args \(robot.running.model.Keyword attribute\), 518](#)  
[ARGUMENT \(robot.parsing.lexer.tokens.END attribute\), 303](#)  
[ARGUMENT \(robot.parsing.lexer.tokens.EOS attribute\), 301](#)  
[ARGUMENT \(robot.parsing.lexer.tokens.Token attribute\), 300](#)  
[argument\\_names \(robot.running.arguments.argumentsparser.attribute\), 494](#)  
[ArgumentCoercer \(class in robot.libraries.Remote\), 94](#)  
[ArgumentConverter \(class in robot.running.arguments.argumentconverter\), 492](#)  
[ArgumentHandler \(class in robot.result.xmllelementhandlers\), 489](#)  
[ArgumentMapper \(class in robot.running.arguments.argumentmapper\), 492](#)  
[ArgumentParser \(class in robot.utils.argumentparser\), 554](#)  
[ArgumentResolver \(class in robot.running.arguments.argumentresolver\), 493](#)  
[Arguments \(class in robot.parsing.model.statements\), 333](#)  
[ARGUMENTS \(robot.parsing.lexer.tokens.END attribute\), 304](#)  
[ARGUMENTS \(robot.parsing.lexer.tokens.EOS attribute\), 301](#)  
[ARGUMENTS \(robot.parsing.lexer.tokens.Token attribute\), 300](#)  
[arguments \(robot.running.userkeywordrunner.EmbeddedArgumentsRunner attribute\), 552](#)  
[arguments \(robot.running.userkeywordrunner.UserKeywordRunner attribute\), 552](#)  
[ArgumentsHandler \(class in robot.result.xmllelementhandlers\), 489](#)  
[ArgumentSpec \(class in robot.running.arguments.argumentspec\), 494](#)  
[ArgumentValidator \(class in robot.running.arguments.argumentvalidator\), 494](#)  
[AS \(robot.parsing.lexer.tokens.END attribute\), 304](#)  
[AS \(robot.parsing.lexer.tokens.EOS attribute\), 301](#)  
[AS \(robot.parsing.lexer.tokens.Token attribute\), 300](#)  
[as\\_dict \(\) \(robot.variables.scopes.GlobalVariables method\), 571](#)  
[as\\_dict \(\) \(robot.variables.scopes.VariableScopes method\), 571](#)  
[as\\_dict \(\) \(robot.variables.store.VariableStore method\), 573](#)  
[as\\_dict \(\) \(robot.variables.variables.Variables method\), 574](#)  
[aspect \(\) \(robot.libraries.dialogs\\_py.InputDialog method\), 137](#)  
[aspect \(\) \(robot.libraries.dialogs\\_py.MessageDialog method\), 123](#)  
[aspect \(\) \(robot.libraries.dialogs\\_py.MultipleSelectionDialog method\), 165](#)  
[aspect \(\) \(robot.libraries.dialogs\\_py.PassFailDialog method\), 179](#)  
[aspect \(\) \(robot.libraries.dialogs\\_py.SelectionDialog method\), 151](#)  
[assert\\_almost\\_equal \(\) \(in module robot.utils.asserts\), 557](#)  
[assert\\_equal \(\) \(in module robot.utils.asserts\), 556](#)  
[assert\\_false \(\) \(in module robot.utils.asserts\), 556](#)  
[assert\\_none \(\) \(in module robot.utils.asserts\), 556](#)  
[assert\\_not\\_almost\\_equal \(\) \(in module robot.utils.asserts\), 557](#)  
[assert\\_not\\_equal \(\) \(in module robot.utils.asserts\), 557](#)

- `assert_not_none()` (in module `robot.utils.asserts`), 556
- `assert_raises()` (in module `robot.utils.asserts`), 556
- `assert_raises_with_msg()` (in module `robot.utils.asserts`), 556
- `assert_true()` (in module `robot.utils.asserts`), 556
- `assign` (`robot.model.keyword.Keyword` attribute), 224
- `ASSIGN` (`robot.parsing.lexer.tokens.END` attribute), 304
- `ASSIGN` (`robot.parsing.lexer.tokens.EOS` attribute), 301
- `ASSIGN` (`robot.parsing.lexer.tokens.Token` attribute), 300
- `assign` (`robot.parsing.model.blocks.If` attribute), 309
- `assign` (`robot.parsing.model.statements.ElseHeader` attribute), 341
- `assign` (`robot.parsing.model.statements.ElseIfHeader` attribute), 340
- `assign` (`robot.parsing.model.statements.IfElseHeader` attribute), 338
- `assign` (`robot.parsing.model.statements.IfHeader` attribute), 339
- `assign` (`robot.parsing.model.statements.InlineIfHeader` attribute), 339
- `assign` (`robot.parsing.model.statements.KeywordCall` attribute), 335
- `assign` (`robot.result.model.Break` attribute), 452
- `assign` (`robot.result.model.Continue` attribute), 450
- `assign` (`robot.result.model.For` attribute), 432
- `assign` (`robot.result.model.ForIteration` attribute), 430
- `assign` (`robot.result.model.If` attribute), 441
- `assign` (`robot.result.model.IfBranch` attribute), 439
- `assign` (`robot.result.model.Keyword` attribute), 455
- `assign` (`robot.result.model.Return` attribute), 448
- `assign` (`robot.result.model.Try` attribute), 445
- `assign` (`robot.result.model.TryBranch` attribute), 443
- `assign` (`robot.result.model.While` attribute), 436
- `assign` (`robot.result.model.WhileIteration` attribute), 434
- `assign` (`robot.result.model.deprecation.DeprecatedAttributesMixin` attribute), 464
- `assign` (`robot.running.model.Keyword` attribute), 518
- `assign()` (`robot.variables.assigner.VariableAssigner` method), 568
- `assigner()` (`robot.variables.assigner.VariableAssignment` method), 568
- `AssignHandler` (class in `robot.result.xml.elementhandlers`), 489
- `AssignmentValidator` (class in `robot.variables.assigner`), 568
- `attribute_escape()` (in module `robot.utils.markuputils`), 564
- `attributes()` (`robot.libraries.dialogs_py.InputDialog` method), 137
- `attributes()` (`robot.libraries.dialogs_py.MessageDialog` method), 123
- `attributes()` (`robot.libraries.dialogs_py.MultipleSelectionDialog` method), 165
- `attributes()` (`robot.libraries.dialogs_py.PassFailDialog` method), 179
- `attributes()` (`robot.libraries.dialogs_py.SelectionDialog` method), 151
- ## B
- `BaseBody` (class in `robot.model.body`), 194
- `BaseParser` (class in `robot.running.builder.parsers`), 505
- `bbox()` (`robot.libraries.dialogs_py.InputDialog` method), 137
- `bbox()` (`robot.libraries.dialogs_py.MessageDialog` method), 123
- `bbox()` (`robot.libraries.dialogs_py.MultipleSelectionDialog` method), 165
- `bbox()` (`robot.libraries.dialogs_py.PassFailDialog` method), 179
- `bbox()` (`robot.libraries.dialogs_py.SelectionDialog` method), 151
- `before` (`robot.variables.search.VariableMatch` attribute), 572
- `bell()` (`robot.libraries.dialogs_py.InputDialog` method), 137
- `bell()` (`robot.libraries.dialogs_py.MessageDialog` method), 123
- `bell()` (`robot.libraries.dialogs_py.MultipleSelectionDialog` method), 165
- `bell()` (`robot.libraries.dialogs_py.PassFailDialog` method), 179
- `bell()` (`robot.libraries.dialogs_py.SelectionDialog` method), 151
- `binary` (`robot.libraries.Remote.ArgumentCoercer` attribute), 94
- `bind()` (`robot.libraries.dialogs_py.InputDialog` method), 137
- `bind()` (`robot.libraries.dialogs_py.MessageDialog` method), 123
- `bind()` (`robot.libraries.dialogs_py.MultipleSelectionDialog` method), 165
- `bind()` (`robot.libraries.dialogs_py.PassFailDialog` method), 179
- `bind()` (`robot.libraries.dialogs_py.SelectionDialog` method), 151
- `bind_all()` (`robot.libraries.dialogs_py.InputDialog` method), 138
- `bind_all()` (`robot.libraries.dialogs_py.MessageDialog` method), 124
- `bind_all()` (`robot.libraries.dialogs_py.MultipleSelectionDialog` method), 166
- `bind_all()` (`robot.libraries.dialogs_py.PassFailDialog` method), 180

- `bind_all()` (*robot.libraries.dialogs\_py.SelectionDialog* method), 152
- `bind_class()` (*robot.libraries.dialogs\_py.InputDialog* method), 138
- `bind_class()` (*robot.libraries.dialogs\_py.MessageDialog* method), 124
- `bind_class()` (*robot.libraries.dialogs\_py.MultipleSelectionDialog* method), 166
- `bind_class()` (*robot.libraries.dialogs\_py.PassFailDialog* method), 180
- `bind_class()` (*robot.libraries.dialogs\_py.SelectionDialog* method), 152
- `bindtags()` (*robot.libraries.dialogs\_py.InputDialog* method), 138
- `bindtags()` (*robot.libraries.dialogs\_py.MessageDialog* method), 124
- `bindtags()` (*robot.libraries.dialogs\_py.MultipleSelectionDialog* method), 166
- `bindtags()` (*robot.libraries.dialogs\_py.PassFailDialog* method), 180
- `bindtags()` (*robot.libraries.dialogs\_py.SelectionDialog* method), 152
- `bit_length()` (*robot.reporting.stringcache.StringIndex* method), 365
- `Block` (class in *robot.parsing.model.blocks*), 306
- `BlockLexer` (class in *robot.parsing.lexer.blocklexers*), 283
- `BlockParser` (class in *robot.parsing.parser.blockparsers*), 353
- `Body` (class in *robot.model.body*), 196
- `Body` (class in *robot.result.model*), 422
- `Body` (class in *robot.running.model*), 516
- `body` (*robot.model.control.For* attribute), 204
- `body` (*robot.model.control.If* attribute), 207
- `body` (*robot.model.control.IfBranch* attribute), 206
- `body` (*robot.model.control.Try* attribute), 210
- `body` (*robot.model.control.TryBranch* attribute), 209
- `body` (*robot.model.control.While* attribute), 205
- `body` (*robot.model.testcase.TestCase* attribute), 248
- `body` (*robot.result.model.Break* attribute), 452
- `body` (*robot.result.model.Continue* attribute), 449
- `body` (*robot.result.model.For* attribute), 431
- `body` (*robot.result.model.ForIteration* attribute), 429
- `body` (*robot.result.model.If* attribute), 441
- `body` (*robot.result.model.IfBranch* attribute), 439
- `body` (*robot.result.model.Keyword* attribute), 454
- `body` (*robot.result.model.Return* attribute), 447
- `body` (*robot.result.model.TestCase* attribute), 458
- `body` (*robot.result.model.Try* attribute), 445
- `body` (*robot.result.model.TryBranch* attribute), 443
- `body` (*robot.result.model.While* attribute), 436
- `body` (*robot.result.model.WhileIteration* attribute), 433
- `body` (*robot.running.model.For* attribute), 520
- `body` (*robot.running.model.If* attribute), 525
- `body` (*robot.running.model.IfBranch* attribute), 523
- `body` (*robot.running.model.TestCase* attribute), 532
- `body` (*robot.running.model.TryBranch* attribute), 527
- `body` (*robot.running.model.TryBranch* attribute), 526
- `body` (*robot.running.model.UserKeyword* attribute), 538
- `body` (*robot.running.model.While* attribute), 522
- `body_class` (*robot.model.control.For* attribute), 203
- `body_class` (*robot.model.control.IfBranch* attribute), 206
- `body_class` (*robot.model.control.TryBranch* attribute), 209
- `body_class` (*robot.model.control.While* attribute), 205
- `body_class` (*robot.model.testcase.TestCase* attribute), 247
- `body_class` (*robot.result.model.Break* attribute), 451
- `body_class` (*robot.result.model.Continue* attribute), 449
- `body_class` (*robot.result.model.For* attribute), 432
- `body_class` (*robot.result.model.ForIteration* attribute), 429
- `body_class` (*robot.result.model.IfBranch* attribute), 438
- `body_class` (*robot.result.model.Keyword* attribute), 454
- `body_class` (*robot.result.model.Return* attribute), 447
- `body_class` (*robot.result.model.TestCase* attribute), 457
- `body_class` (*robot.result.model.TryBranch* attribute), 442
- `body_class` (*robot.result.model.While* attribute), 436
- `body_class` (*robot.result.model.WhileIteration* attribute), 433
- `body_class` (*robot.running.model.For* attribute), 520
- `body_class` (*robot.running.model.IfBranch* attribute), 523
- `body_class` (*robot.running.model.TestCase* attribute), 532
- `body_class` (*robot.running.model.TryBranch* attribute), 525
- `body_class` (*robot.running.model.While* attribute), 521
- `BodyItem` (class in *robot.model.body*), 193
- `BodyRunner` (class in *robot.running.bodyrunner*), 511
- `BooleanConverter` (class in *robot.running.arguments.typeconverters*), 496
- `branch_class` (*robot.model.body.Branches* attribute), 197
- `branch_class` (*robot.model.control.If* attribute), 207
- `branch_class` (*robot.model.control.Try* attribute), 210
- `branch_class` (*robot.result.model.Branches* attribute), 423
- `branch_class` (*robot.result.model.If* attribute), 440

- [branch\\_class \(robot.result.model.Try attribute\), 444](#)
- [branch\\_class \(robot.running.model.If attribute\), 524](#)
- [branch\\_class \(robot.running.model.Try attribute\), 527](#)
- [Branches \(class in robot.model.body\), 197](#)
- [Branches \(class in robot.result.model\), 423](#)
- [branches\\_class \(robot.model.control.If attribute\), 207](#)
- [branches\\_class \(robot.model.control.Try attribute\), 210](#)
- [branches\\_class \(robot.result.model.If attribute\), 440](#)
- [branches\\_class \(robot.result.model.Try attribute\), 445](#)
- [branches\\_class \(robot.running.model.If attribute\), 525](#)
- [branches\\_class \(robot.running.model.Try attribute\), 527](#)
- [BranchHandler \(class in robot.result.xmlélémenthandlers\), 485](#)
- [Break \(class in robot.model.control\), 214](#)
- [Break \(class in robot.parsing.model.statements\), 349](#)
- [Break \(class in robot.result.model\), 451](#)
- [Break \(class in robot.running.model\), 531](#)
- [BREAK \(robot.model.body.BodyItem attribute\), 193](#)
- [BREAK \(robot.model.control.Break attribute\), 214](#)
- [BREAK \(robot.model.control.Continue attribute\), 213](#)
- [BREAK \(robot.model.control.For attribute\), 204](#)
- [BREAK \(robot.model.control.If attribute\), 208](#)
- [BREAK \(robot.model.control.IfBranch attribute\), 206](#)
- [BREAK \(robot.model.control.Return attribute\), 212](#)
- [BREAK \(robot.model.control.Try attribute\), 210](#)
- [BREAK \(robot.model.control.TryBranch attribute\), 209](#)
- [BREAK \(robot.model.control.While attribute\), 205](#)
- [BREAK \(robot.model.keyword.Keyword attribute\), 225](#)
- [BREAK \(robot.model.message.Message attribute\), 228](#)
- [BREAK \(robot.output.loggerhelper.Message attribute\), 274](#)
- [BREAK \(robot.parsing.lexer.tokens.END attribute\), 304](#)
- [BREAK \(robot.parsing.lexer.tokens.EOS attribute\), 301](#)
- [BREAK \(robot.parsing.lexer.tokens.Token attribute\), 301](#)
- [BREAK \(robot.result.model.Break attribute\), 452](#)
- [BREAK \(robot.result.model.Continue attribute\), 449](#)
- [BREAK \(robot.result.model.For attribute\), 431](#)
- [BREAK \(robot.result.model.ForIteration attribute\), 429](#)
- [BREAK \(robot.result.model.If attribute\), 440](#)
- [BREAK \(robot.result.model.IfBranch attribute\), 438](#)
- [BREAK \(robot.result.model.Keyword attribute\), 455](#)
- [BREAK \(robot.result.model.Message attribute\), 427](#)
- [BREAK \(robot.result.model.Return attribute\), 447](#)
- [BREAK \(robot.result.model.Try attribute\), 445](#)
- [BREAK \(robot.result.model.TryBranch attribute\), 442](#)
- [BREAK \(robot.result.model.While attribute\), 436](#)
- [BREAK \(robot.result.model.WhileIteration attribute\), 434](#)
- [BREAK \(robot.running.model.Break attribute\), 531](#)
- [BREAK \(robot.running.model.Continue attribute\), 530](#)
- [BREAK \(robot.running.model.For attribute\), 520](#)
- [BREAK \(robot.running.model.If attribute\), 524](#)
- [BREAK \(robot.running.model.IfBranch attribute\), 523](#)
- [BREAK \(robot.running.model.Keyword attribute\), 518](#)
- [BREAK \(robot.running.model.Return attribute\), 528](#)
- [BREAK \(robot.running.model.Try attribute\), 527](#)
- [BREAK \(robot.running.model.TryBranch attribute\), 526](#)
- [BREAK \(robot.running.model.While attribute\), 521](#)
- [break\\_class \(robot.model.body.BaseBody attribute\), 194](#)
- [break\\_class \(robot.model.body.Body attribute\), 196](#)
- [break\\_class \(robot.model.body.Branches attribute\), 197](#)
- [break\\_class \(robot.result.model.Body attribute\), 422](#)
- [break\\_class \(robot.result.model.Branches attribute\), 423](#)
- [break\\_class \(robot.result.model.Iterations attribute\), 425](#)
- [break\\_class \(robot.running.model.Body attribute\), 516](#)
- [BreakHandler \(class in robot.result.xmlélémenthandlers\), 486](#)
- [BreakLexer \(class in robot.parsing.lexer.statementslexers\), 299](#)
- [BreakLoop, 579](#)
- [build\(\) \(robot.libdocpkg.jsonbuilder.JsonDocBuilder method\), 37](#)
- [build\(\) \(robot.libdocpkg.robotbuilder.LibraryDocBuilder method\), 38](#)
- [build\(\) \(robot.libdocpkg.robotbuilder.ResourceDocBuilder method\), 38](#)
- [build\(\) \(robot.libdocpkg.xmlbuilder.XmlDocBuilder method\), 38](#)
- [build\(\) \(robot.parsing.suitestructure.SuiteStructureBuilder method\), 357](#)
- [build\(\) \(robot.reporting.jsmodelbuilders.ErrorMessageBuilder method\), 359](#)
- [build\(\) \(robot.reporting.jsmodelbuilders.ErrorsBuilder method\), 358](#)
- [build\(\) \(robot.reporting.jsmodelbuilders.KeywordBuilder method\), 358](#)
- [build\(\) \(robot.reporting.jsmodelbuilders.MessageBuilder method\), 358](#)
- [build\(\) \(robot.reporting.jsmodelbuilders.StatisticsBuilder method\), 358](#)
- [build\(\) \(robot.reporting.jsmodelbuilders.SuiteBuilder method\), 358](#)
- [build\(\) \(robot.reporting.jsmodelbuilders.TestBuilder method\), 358](#)
- [build\(\) \(robot.result.resultbuilder.ExecutionResultBuilder method\), 465](#)
- [build\(\) \(robot.running.builder.builders.ResourceFileBuilder](#)



- method), 504
- build() (robot.running.builder.builders.TestSuiteBuilder method), 504
- build() (robot.running.builder.transformers.ForBuilder method), 508
- build() (robot.running.builder.transformers.IfBuilder method), 509
- build() (robot.running.builder.transformers.TryBuilder method), 509
- build() (robot.running.builder.transformers.WhileBuilder method), 509
- build\_from() (robot.reporting.jsmodelbuilders JsModelBuilder method), 358
- build\_from\_dict() (robot.libdocpkg.jsonbuilder.JsonDocBuilder method), 37
- build\_keyword() (robot.libdocpkg.robotbuilder.KeywordDocBuilder method), 38
- build\_keyword() (robot.reporting.jsmodelbuilders.KeywordBuilder method), 358
- build\_keywords() (robot.libdocpkg.robotbuilder.KeywordDocBuilder method), 38
- build\_suite() (robot.running.builder.parsers.RestParser method), 505
- build\_suite() (robot.running.builder.parsers.RobotParser method), 505
- BuiltIn (class in robot.libraries.BuiltIn), 39
- by\_method\_name() (robot.output.listenerarguments.EndKeywordArguments class method), 271
- by\_method\_name() (robot.output.listenerarguments.EndSuiteArguments class method), 271
- by\_method\_name() (robot.output.listenerarguments.EndTestArguments class method), 271
- by\_method\_name() (robot.output.listenerarguments.ListenerArguments class method), 271
- by\_method\_name() (robot.output.listenerarguments.MessageArguments class method), 271
- by\_method\_name() (robot.output.listenerarguments.StartKeywordArguments class method), 271
- by\_method\_name() (robot.output.listenerarguments.StartSuiteArguments class method), 271
- by\_method\_name() (robot.output.listenerarguments.StartTestArguments class method), 271
- ByNameKeywordRemover (class in robot.result.keywordremover), 388
- ByPathImporter (class in robot.utils.importer), 563
- ByTagKeywordRemover (class in robot.result.keywordremover), 392
- ByteArrayConverter (class in robot.running.arguments.typeconverters), 498
- BytesConverter (class in robot.running.arguments.typeconverters), 498
- C
- cache\_only (robot.output.logger.Logger attribute), 273
- call\_method() (robot.libraries.BuiltIn.BuiltIn method), 42
- called (robot.output.listenermethods.ListenerMethod attribute), 272
- can\_continue() (robot.errors.BreakLoop method), 579
- can\_continue() (robot.errors.ContinueLoop method), 578
- can\_continue() (robot.errors.ExecutionFailed method), 576
- can\_continue() (robot.errors.ExecutionFailures method), 577
- can\_continue() (robot.errors.ExecutionPassed method), 578
- can\_continue() (robot.errors.ExecutionStatus method), 576
- can\_continue() (robot.errors.HandlerExecutionFailed method), 577
- can\_continue() (robot.errors.PassExecution method), 578
- can\_continue() (robot.errors.ReturnFromKeyword method), 579
- can\_continue() (robot.errors.UserKeywordExecutionFailed method), 577
- concatenate() (robot.libraries.BuiltIn.BuiltIn method), 42
- create() (robot.libraries.dialogs\_py.InputDialog method), 138
- create() (robot.libraries.dialogs\_py.MessageDialog method), 124
- create() (robot.libraries.dialogs\_py.MultipleSelectionDialog method), 166
- create() (robot.libraries.dialogs\_py.PassFailDialog method), 180
- create() (robot.libraries.dialogs\_py.SelectionDialog method), 152
- check\_expansion() (robot.reporting.jsbuildingcontext JsBuildingContext method), 357
- child() (robot.libraries.XML.Location method), 123
- children (robot.result.model.Keyword attribute), 454
- children (robot.result.xmllelementhandlers.ArgumentHandler attribute), 490
- children (robot.result.xmllelementhandlers.ArgumentsHandler attribute), 489
- children (robot.result.xmllelementhandlers.AssignHandler attribute), 489
- children (robot.result.xmllelementhandlers.BranchHandler attribute), 485
- children (robot.result.xmllelementhandlers.BreakHandler attribute), 486

`children (robot.result.xml_element_handlers.ContinueHandler attribute)`, 486

`children (robot.result.xml_element_handlers.DocHandler attribute)`, 487

`children (robot.result.xml_element_handlers.ElementHandler attribute)`, 483

`children (robot.result.xml_element_handlers.ErrorMessageHandler attribute)`, 490

`children (robot.result.xml_element_handlers.ErrorsHandler attribute)`, 490

`children (robot.result.xml_element_handlers.ForHandler attribute)`, 484

`children (robot.result.xml_element_handlers.IfHandler attribute)`, 485

`children (robot.result.xml_element_handlers.IterationHandler attribute)`, 485

`children (robot.result.xml_element_handlers.KeywordHandler attribute)`, 484

`children (robot.result.xml_element_handlers.MessageHandler attribute)`, 487

`children (robot.result.xml_element_handlers.MetadataHandler attribute)`, 487

`children (robot.result.xml_element_handlers.MetadataItemHandler attribute)`, 488

`children (robot.result.xml_element_handlers.MetaHandler attribute)`, 488

`children (robot.result.xml_element_handlers.PatternHandler attribute)`, 486

`children (robot.result.xml_element_handlers.ReturnHandler attribute)`, 486

`children (robot.result.xml_element_handlers.RobotHandler attribute)`, 483

`children (robot.result.xml_element_handlers.RootHandler attribute)`, 483

`children (robot.result.xml_element_handlers.StatisticsHandler attribute)`, 490

`children (robot.result.xml_element_handlers.StatusHandler attribute)`, 487

`children (robot.result.xml_element_handlers.SuiteHandler attribute)`, 483

`children (robot.result.xml_element_handlers.TagHandler attribute)`, 488

`children (robot.result.xml_element_handlers.TagsHandler attribute)`, 488

`children (robot.result.xml_element_handlers.TestHandler attribute)`, 484

`children (robot.result.xml_element_handlers.TimeoutHandler attribute)`, 489

`children (robot.result.xml_element_handlers.TryHandler attribute)`, 485

`children (robot.result.xml_element_handlers.ValueHandler attribute)`, 490

`children (robot.result.xml_element_handlers.VarHandler attribute)`, 489

`children (robot.result.xml_element_handlers.WhileHandler attribute)`, 484

`clear () (robot.model.body.BaseBody method)`, 195

`clear () (robot.model.body.Body method)`, 196

`clear () (robot.model.body.Branches method)`, 197

`clear () (robot.model.itemlist.ItemList method)`, 224

`clear () (robot.model.keyword.Keywords method)`, 227

`clear () (robot.model.message.Messages method)`, 229

`clear () (robot.model.metadata.Metadata method)`, 229

`clear () (robot.model.testcase.TestCases method)`, 249

`clear () (robot.model.testsuite.TestSuites method)`, 252

`clear () (robot.result.model.Body method)`, 422

`clear () (robot.result.model.Branches method)`, 423

`clear () (robot.result.model.Iterations method)`, 425

`clear () (robot.running.model.Body method)`, 516

`clear () (robot.running.model.Imports method)`, 539

`clear () (robot.utils.dotdict.DotDict method)`, 558

`clear () (robot.utils.normalizing.NormalizedDict method)`, 566

`clear () (robot.variables.scopes.GlobalVariables method)`, 571

`clear () (robot.variables.store.VariableStore method)`, 573

`clear () (robot.variables.variables.Variables method)`, 574

`clear_element () (robot.libraries.XML.XML method)`, 121

`client () (robot.libraries.dialogs_py.InputDialog method)`, 138

`client () (robot.libraries.dialogs_py.MessageDialog method)`, 124

`client () (robot.libraries.dialogs_py.MultipleSelectionDialog method)`, 166

`client () (robot.libraries.dialogs_py.PassFailDialog method)`, 180

`client () (robot.libraries.dialogs_py.SelectionDialog method)`, 152

`clipboard_append () (robot.libraries.dialogs_py.InputDialog method)`, 138

`clipboard_append () (robot.libraries.dialogs_py.MessageDialog method)`, 124

`clipboard_append () (robot.libraries.dialogs_py.MultipleSelectionDialog method)`, 166

`clipboard_append () (robot.libraries.dialogs_py.PassFailDialog method)`, 180

`clipboard_append () (robot.libraries.dialogs_py.SelectionDialog method)`, 152

`clipboard_clear ()`

(*robot.libraries.dialogs\_py.InputDialog*  
*method*), 138  
*clipboard\_clear()*  
 (*robot.libraries.dialogs\_py.MessageDialog*  
*method*), 124  
*clipboard\_clear()*  
 (*robot.libraries.dialogs\_py.MultipleSelectionDialog*  
*method*), 166  
*clipboard\_clear()*  
 (*robot.libraries.dialogs\_py.PassFailDialog*  
*method*), 180  
*clipboard\_clear()*  
 (*robot.libraries.dialogs\_py.SelectionDialog*  
*method*), 152  
*clipboard\_get()* (*robot.libraries.dialogs\_py.InputDialog*  
*method*), 138  
*clipboard\_get()* (*robot.libraries.dialogs\_py.MessageDialog*  
*method*), 124  
*clipboard\_get()* (*robot.libraries.dialogs\_py.MultipleSelectionDialog*  
*method*), 166  
*clipboard\_get()* (*robot.libraries.dialogs\_py.PassFailDialog*  
*method*), 180  
*clipboard\_get()* (*robot.libraries.dialogs\_py.SelectionDialog*  
*method*), 152  
*close()* (*robot.libraries.Remote.TimeoutHTTPSTransport*  
*method*), 95  
*close()* (*robot.libraries.Remote.TimeoutHTTPSTransport*  
*method*), 94  
*close()* (*robot.libraries.Telnet.TelnetConnection*  
*method*), 110  
*close()* (*robot.output.filelogger.FileLogger* *method*),  
 270  
*close()* (*robot.output.logger.Logger* *method*), 273  
*close()* (*robot.output.output.Output* *method*), 276  
*close()* (*robot.output.pyloggingconf.RobotHandler*  
*method*), 277  
*close()* (*robot.output.xmllogger.XmlLogger* *method*),  
 278  
*close()* (*robot.reporting.outputwriter.OutputWriter*  
*method*), 359  
*close()* (*robot.utils.application.DefaultLogger*  
*method*), 554  
*close()* (*robot.utils.markupwriters.HtmlWriter*  
*method*), 564  
*close()* (*robot.utils.markupwriters.NullMarkupWriter*  
*method*), 565  
*close()* (*robot.utils.markupwriters.XmlWriter*  
*method*), 564  
*close\_all()* (*robot.utils.connectioncache.ConnectionCache*  
*method*), 558  
*close\_all\_connections()*  
 (*robot.libraries.Telnet.Telnet* *method*), 106  
*close\_connection()*  
 (*robot.libraries.Telnet.TelnetConnection*  
*method*), 108  
*close\_global\_library\_listeners()*  
 (*robot.running.importer.Importer* *method*),  
 514  
*close\_streams()* (*robot.libraries.Process.ExecutionResult*  
*method*), 93  
*cmdline2list()* (*in* *robot.utils.argumentparser* module  
*robot.utils.argumentparser*), 554  
*coerce()* (*robot.libraries.Remote.ArgumentCoercer*  
*method*), 94  
*col\_offset* (*robot.parsing.lexer.tokens.END* *at-*  
*tribute*), 305  
*col\_offset* (*robot.parsing.lexer.tokens.EOS* *at-*  
*tribute*), 303  
*col\_offset* (*robot.parsing.lexer.tokens.Token* *at-*  
*tribute*), 301  
*col\_offset* (*robot.parsing.model.blocks.Block* *at-*  
*tribute*), 306  
*col\_offset* (*robot.parsing.model.blocks.CommentSection*  
*attribute*), 308  
*col\_offset* (*robot.parsing.model.blocks.File* *at-*  
*tribute*), 306  
*col\_offset* (*robot.parsing.model.blocks.For* *at-*  
*tribute*), 309  
*col\_offset* (*robot.parsing.model.blocks.HeaderAndBody*  
*attribute*), 306  
*col\_offset* (*robot.parsing.model.blocks.If* *attribute*),  
 309  
*col\_offset* (*robot.parsing.model.blocks.Keyword* *at-*  
*tribute*), 308  
*col\_offset* (*robot.parsing.model.blocks.KeywordSection*  
*attribute*), 307  
*col\_offset* (*robot.parsing.model.blocks.Section* *at-*  
*tribute*), 306  
*col\_offset* (*robot.parsing.model.blocks.SettingSection*  
*attribute*), 307  
*col\_offset* (*robot.parsing.model.blocks.TestCase* *at-*  
*tribute*), 308  
*col\_offset* (*robot.parsing.model.blocks.TestCaseSection*  
*attribute*), 307  
*col\_offset* (*robot.parsing.model.blocks.Try* *at-*  
*tribute*), 309  
*col\_offset* (*robot.parsing.model.blocks.VariableSection*  
*attribute*), 307  
*col\_offset* (*robot.parsing.model.blocks.While* *at-*  
*tribute*), 310  
*col\_offset* (*robot.parsing.model.statements.Arguments*  
*attribute*), 334  
*col\_offset* (*robot.parsing.model.statements.Break*  
*attribute*), 349  
*col\_offset* (*robot.parsing.model.statements.Comment*  
*attribute*), 350  
*col\_offset* (*robot.parsing.model.statements.Continue*  
*attribute*), 348

`col_offset (robot.parsing.model.statements.DefaultTags attribute), 321`

`col_offset (robot.parsing.model.statements.Documentation attribute), 319`

`col_offset (robot.parsing.model.statements.DocumentationOrOffset attribute), 312`

`col_offset (robot.parsing.model.statements.ElseHeader attribute), 341`

`col_offset (robot.parsing.model.statements.ElseIfHeader attribute), 340`

`col_offset (robot.parsing.model.statements.EmptyLine attribute), 352`

`col_offset (robot.parsing.model.statements.End attribute), 345`

`col_offset (robot.parsing.model.statements.Error attribute), 351`

`col_offset (robot.parsing.model.statements.ExceptHeader attribute), 344`

`col_offset (robot.parsing.model.statements.FinallyHeader attribute), 344`

`col_offset (robot.parsing.model.statements.Fixture attribute), 314`

`col_offset (robot.parsing.model.statements.ForceTags attribute), 320`

`col_offset (robot.parsing.model.statements.ForHeader attribute), 337`

`col_offset (robot.parsing.model.statements.IfElseHeader attribute), 338`

`col_offset (robot.parsing.model.statements.IfHeader attribute), 339`

`col_offset (robot.parsing.model.statements.InlineIfHeader attribute), 339`

`col_offset (robot.parsing.model.statements.KeywordCall attribute), 335`

`col_offset (robot.parsing.model.statements.KeywordName attribute), 329`

`col_offset (robot.parsing.model.statements.LibraryImport attribute), 316`

`col_offset (robot.parsing.model.statements.LoopControl attribute), 348`

`col_offset (robot.parsing.model.statements.Metadata attribute), 320`

`col_offset (robot.parsing.model.statements.MultiValue attribute), 313`

`col_offset (robot.parsing.model.statements.NoArgumentHeader attribute), 342`

`col_offset (robot.parsing.model.statements.ResourceImport attribute), 317`

`col_offset (robot.parsing.model.statements.Return attribute), 334`

`col_offset (robot.parsing.model.statements.ReturnStatement attribute), 347`

`col_offset (robot.parsing.model.statements.SectionHeader attribute), 315`

`col_offset (robot.parsing.model.statements.Setup attribute), 330`

`col_offset (robot.parsing.model.statements.SingleValue attribute), 313`

`col_offset (robot.parsing.model.statements.Statement attribute), 311`

`col_offset (robot.parsing.model.statements.SuiteSetup attribute), 322`

`col_offset (robot.parsing.model.statements.SuiteTeardown attribute), 323`

`col_offset (robot.parsing.model.statements.Tags attribute), 331`

`col_offset (robot.parsing.model.statements.Teardown attribute), 330`

`col_offset (robot.parsing.model.statements.Template attribute), 332`

`col_offset (robot.parsing.model.statements.TemplateArguments attribute), 336`

`col_offset (robot.parsing.model.statements.TestCaseName attribute), 328`

`col_offset (robot.parsing.model.statements.TestSetup attribute), 324`

`col_offset (robot.parsing.model.statements.TestTeardown attribute), 325`

`col_offset (robot.parsing.model.statements.TestTemplate attribute), 325`

`col_offset (robot.parsing.model.statements.TestTimeout attribute), 326`

`col_offset (robot.parsing.model.statements.Timeout attribute), 333`

`col_offset (robot.parsing.model.statements.TryHeader attribute), 343`

`col_offset (robot.parsing.model.statements.Variable attribute), 327`

`col_offset (robot.parsing.model.statements.VariablesImport attribute), 318`

`col_offset (robot.parsing.model.statements.WhileHeader attribute), 346`

`collections (class in robot.libraries.Collections), 65`

`colormapwindows ()`

`colormapwindows () (robot.libraries.dialogs_py.InputDialog method), 138`

`colormapwindows ()`

`colormapwindows () (robot.libraries.dialogs_py.MessageDialog method), 125`

`colormapwindows ()`

`colormapwindows () (robot.libraries.dialogs_py.MultipleSelectionDialog method), 166`

`colormapwindows ()`

`colormapwindows () (robot.libraries.dialogs_py.PassFailDialog method), 180`

`colormapwindows ()`

`colormapwindows () (robot.libraries.dialogs_py.SelectionDialog method), 152`



- `columnconfigure()` (`robot.libraries.dialogs_py.InputDialog` method), 139
- `columnconfigure()` (`robot.libraries.dialogs_py.MessageDialog` method), 125
- `columnconfigure()` (`robot.libraries.dialogs_py.MultipleSelectionDialog` method), 167
- `columnconfigure()` (`robot.libraries.dialogs_py.PassFailDialog` method), 181
- `columnconfigure()` (`robot.libraries.dialogs_py.SelectionDialog` method), 153
- `combine_lists()` (`robot.libraries.Collections.Collections` method), 67
- `combined` (`robot.model.stats.TagStat` attribute), 241
- `combined` (`robot.model.tagstatistics.TagStatistics` attribute), 247
- `CombinedConverter` (class in `robot.running.arguments.typeconverters`), 502
- `CombinedResult` (class in `robot.result.executionresult`), 378
- `CombinedTagStat` (class in `robot.model.stats`), 241
- `command()` (`robot.libraries.dialogs_py.InputDialog` method), 139
- `command()` (`robot.libraries.dialogs_py.MessageDialog` method), 125
- `command()` (`robot.libraries.dialogs_py.MultipleSelectionDialog` method), 167
- `command()` (`robot.libraries.dialogs_py.PassFailDialog` method), 181
- `command()` (`robot.libraries.dialogs_py.SelectionDialog` method), 153
- `Comment` (class in `robot.parsing.model.statements`), 350
- `COMMENT` (`robot.parsing.lexer.tokens.END` attribute), 304
- `COMMENT` (`robot.parsing.lexer.tokens.EOS` attribute), 301
- `COMMENT` (`robot.parsing.lexer.tokens.Token` attribute), 301
- `comment()` (`robot.libraries.BuiltIn.BuiltIn` method), 42
- `comment()` (`robot.parsing.lexer.sections.InitFileSections` method), 291
- `comment()` (`robot.parsing.lexer.sections.ResourceFileSections` method), 291
- `comment()` (`robot.parsing.lexer.sections.Sections` method), 291
- `comment()` (`robot.parsing.lexer.sections.TestCaseFileSections` method), 291
- `COMMENT_HEADER` (`robot.parsing.lexer.tokens.END` attribute), 304
- `COMMENT_HEADER` (`robot.parsing.lexer.tokens.EOS` attribute), 302
- `COMMENT_HEADER` (`robot.parsing.lexer.tokens.Token` attribute), 299
- `comment_markers` (`robot.parsing.lexer.sections.InitFileSections` attribute), 291
- `comment_markers` (`robot.parsing.lexer.sections.ResourceFileSections` attribute), 291
- `comment_markers` (`robot.parsing.lexer.sections.Sections` attribute), 290
- `comment_markers` (`robot.parsing.lexer.sections.TestCaseFileSections` attribute), 291
- `comment_section()` (`robot.parsing.lexer.context.FileContext` method), 288
- `comment_section()` (`robot.parsing.lexer.context.InitFileContext` method), 289
- `comment_section()` (`robot.parsing.lexer.context.ResourceFileContext` method), 289
- `comment_section()` (`robot.parsing.lexer.context.TestCaseFileContext` method), 288
- `CommentLexer` (class in `robot.parsing.lexer.statementslexers`), 295
- `CommentSection` (class in `robot.parsing.model.blocks`), 308
- `CommentSectionHeaderLexer` (class in `robot.parsing.lexer.statementslexers`), 295
- `CommentSectionLexer` (class in `robot.parsing.lexer.blocklexers`), 285
- `CommentSectionParser` (class in `robot.parsing.parser.fileparser`), 355
- `compare()` (`robot.libraries.XML.ElementComparator` method), 122
- `compress_text()` (in module `robot.utils.compress`), 557
- `condition` (`robot.model.control.IfBranch` attribute), 206
- `condition` (`robot.model.control.While` attribute), 205
- `condition` (`robot.parsing.model.blocks.If` attribute), 309
- `condition` (`robot.parsing.model.blocks.While` attribute), 310
- `condition` (`robot.parsing.model.statements.ElseHeader` attribute), 341
- `condition` (`robot.parsing.model.statements.ElseIfHeader` attribute), 340
- `condition` (`robot.parsing.model.statements.IfElseHeader` attribute), 338
- `condition` (`robot.parsing.model.statements.IfHeader` attribute), 339
- `condition` (`robot.parsing.model.statements.InlineIfHeader` attribute), 339

- attribute*), 339
- condition (robot.parsing.model.statements.WhileHeader attribute)*, 346
- condition (robot.result.model.IfBranch attribute)*, 439
- condition (robot.result.model.While attribute)*, 436
- condition (robot.running.model.IfBranch attribute)*, 523
- condition (robot.running.model.While attribute)*, 522
- config() (robot.libraries.dialogs\_py.InputDialog method)*, 139
- config() (robot.libraries.dialogs\_py.MessageDialog method)*, 125
- config() (robot.libraries.dialogs\_py.MultipleSelectionDialog method)*, 167
- config() (robot.libraries.dialogs\_py.PassFailDialog method)*, 181
- config() (robot.libraries.dialogs\_py.SelectionDialog method)*, 153
- config() (robot.model.body.BodyItem method)*, 193
- config() (robot.model.control.Break method)*, 214
- config() (robot.model.control.Continue method)*, 213
- config() (robot.model.control.For method)*, 204
- config() (robot.model.control.If method)*, 208
- config() (robot.model.control.IfBranch method)*, 207
- config() (robot.model.control.Return method)*, 212
- config() (robot.model.control.Try method)*, 211
- config() (robot.model.control.TryBranch method)*, 209
- config() (robot.model.control.While method)*, 205
- config() (robot.model.keyword.Keyword method)*, 226
- config() (robot.model.message.Message method)*, 228
- config() (robot.model.modelobject.ModelObject method)*, 230
- config() (robot.model.testcase.TestCase method)*, 249
- config() (robot.model.testsuite.TestSuite method)*, 252
- config() (robot.output.loggerhelper.Message method)*, 275
- config() (robot.result.model.Break method)*, 452
- config() (robot.result.model.Continue method)*, 450
- config() (robot.result.model.For method)*, 432
- config() (robot.result.model.ForIteration method)*, 430
- config() (robot.result.model.If method)*, 441
- config() (robot.result.model.IfBranch method)*, 439
- config() (robot.result.model.Keyword method)*, 455
- config() (robot.result.model.Message method)*, 427
- config() (robot.result.model.Return method)*, 448
- config() (robot.result.model.TestCase method)*, 458
- config() (robot.result.model.TestSuite method)*, 460
- config() (robot.result.model.Try method)*, 445
- config() (robot.result.model.TryBranch method)*, 443
- config() (robot.result.model.While method)*, 436
- config() (robot.result.model.WhileIteration method)*, 434
- config() (robot.running.model.Break method)*, 532
- config() (robot.running.model.Continue method)*, 530
- config() (robot.running.model.For method)*, 520
- config() (robot.running.model.If method)*, 525
- config() (robot.running.model.IfBranch method)*, 523
- config() (robot.running.model.Keyword method)*, 518
- config() (robot.running.model.Return method)*, 529
- config() (robot.running.model.TestCase method)*, 533
- config() (robot.running.model.TestSuite method)*, 536
- config() (robot.running.model.Try method)*, 528
- config() (robot.running.model.TryBranch method)*, 526
- config() (robot.running.model.While method)*, 522
- configure() (robot.libraries.dialogs\_py.InputDialog method)*, 139
- configure() (robot.libraries.dialogs\_py.MessageDialog method)*, 125
- configure() (robot.libraries.dialogs\_py.MultipleSelectionDialog method)*, 167
- configure() (robot.libraries.dialogs\_py.PassFailDialog method)*, 181
- configure() (robot.libraries.dialogs\_py.SelectionDialog method)*, 153
- configure() (robot.model.testsuite.TestSuite method)*, 252
- configure() (robot.result.executionresult.CombinedResult method)*, 378
- configure() (robot.result.executionresult.Result method)*, 377
- configure() (robot.result.model.TestSuite method)*, 463
- configure() (robot.running.model.TestSuite method)*, 535
- conjugate() (robot.reporting.stringcache.StringIndex method)*, 365
- ConnectionCache (class in robot.utils.connectioncache)*, 557
- console() (in module robot.api.logger)*, 15
- console() (in module robot.output.librarylogger)*, 271
- console() (robot.libdoc.LibDoc method)*, 580
- console() (robot.rebot.Rebot method)*, 582
- console() (robot.run.RobotFramework method)*, 583
- console() (robot.testdoc.TestDoc method)*, 585
- console() (robot.utils.application.Application method)*, 553
- console\_colors (robot.conf.settings.RebotSettings attribute)*, 33
- console\_colors (robot.conf.settings.RobotSettings attribute)*, 32
- console\_decode() (in module robot.utils.encoding)*, 559
- console\_encode() (in module robot.utils.encoding)*, 559
- console\_markers (robot.conf.settings.RobotSettings*

- [attribute](#)), 32
- [console\\_output\\_config](#)
  - ([robot.conf.settings.RobotSettings](#) attribute), 33
- [console\\_output\\_config](#)
  - ([robot.conf.settings.RobotSettings](#) attribute), 32
- [console\\_type](#) ([robot.conf.settings.RobotSettings](#) attribute), 32
- [console\\_width](#) ([robot.conf.settings.RobotSettings](#) attribute), 32
- [ConsoleOutput\(\)](#) (in module [robot.output.console](#)), 263
- [ConsoleViewer](#) (class in [robot.libdocpkg.consoleviewer](#)), 36
- [contains\\_variable\(\)](#) (in module [robot.variables.search](#)), 572
- [content\(\)](#) ([robot.utils.markupwriters.HtmlWriter](#) method), 564
- [content\(\)](#) ([robot.utils.markupwriters.NullMarkupWriter](#) method), 565
- [content\(\)](#) ([robot.utils.markupwriters.XmlWriter](#) method), 565
- [ContinuableFailure](#), 12
- [CONTINUATION](#) ([robot.parsing.lexer.tokens.END](#) attribute), 304
- [CONTINUATION](#) ([robot.parsing.lexer.tokens.EOS](#) attribute), 302
- [CONTINUATION](#) ([robot.parsing.lexer.tokens.Token](#) attribute), 301
- [Continue](#) (class in [robot.model.control](#)), 213
- [Continue](#) (class in [robot.parsing.model.statements](#)), 348
- [Continue](#) (class in [robot.result.model](#)), 449
- [Continue](#) (class in [robot.running.model](#)), 530
- [CONTINUE](#) ([robot.model.body.BodyItem](#) attribute), 193
- [CONTINUE](#) ([robot.model.control.Break](#) attribute), 214
- [CONTINUE](#) ([robot.model.control.Continue](#) attribute), 213
- [CONTINUE](#) ([robot.model.control.For](#) attribute), 204
- [CONTINUE](#) ([robot.model.control.If](#) attribute), 208
- [CONTINUE](#) ([robot.model.control.IfBranch](#) attribute), 206
- [CONTINUE](#) ([robot.model.control.Return](#) attribute), 212
- [CONTINUE](#) ([robot.model.control.Try](#) attribute), 210
- [CONTINUE](#) ([robot.model.control.TryBranch](#) attribute), 209
- [CONTINUE](#) ([robot.model.control.While](#) attribute), 205
- [CONTINUE](#) ([robot.model.keyword.Keyword](#) attribute), 225
- [CONTINUE](#) ([robot.model.message.Message](#) attribute), 228
- [CONTINUE](#) ([robot.output.loggerhelper.Message](#) attribute), 274
- [CONTINUE](#) ([robot.parsing.lexer.tokens.END](#) attribute), 304
- [CONTINUE](#) ([robot.parsing.lexer.tokens.EOS](#) attribute), 302
- [CONTINUE](#) ([robot.parsing.lexer.tokens.Token](#) attribute), 301
- [CONTINUE](#) ([robot.result.model.Break](#) attribute), 452
- [CONTINUE](#) ([robot.result.model.Continue](#) attribute), 450
- [CONTINUE](#) ([robot.result.model.For](#) attribute), 431
- [CONTINUE](#) ([robot.result.model.ForIteration](#) attribute), 429
- [CONTINUE](#) ([robot.result.model.If](#) attribute), 440
- [CONTINUE](#) ([robot.result.model.IfBranch](#) attribute), 438
- [CONTINUE](#) ([robot.result.model.Keyword](#) attribute), 455
- [CONTINUE](#) ([robot.result.model.Message](#) attribute), 427
- [CONTINUE](#) ([robot.result.model.Return](#) attribute), 447
- [CONTINUE](#) ([robot.result.model.Try](#) attribute), 445
- [CONTINUE](#) ([robot.result.model.TryBranch](#) attribute), 442
- [CONTINUE](#) ([robot.result.model.While](#) attribute), 436
- [CONTINUE](#) ([robot.result.model.WhileIteration](#) attribute), 434
- [CONTINUE](#) ([robot.running.model.Break](#) attribute), 531
- [CONTINUE](#) ([robot.running.model.Continue](#) attribute), 530
- [CONTINUE](#) ([robot.running.model.For](#) attribute), 520
- [CONTINUE](#) ([robot.running.model.If](#) attribute), 524
- [CONTINUE](#) ([robot.running.model.IfBranch](#) attribute), 523
- [CONTINUE](#) ([robot.running.model.Keyword](#) attribute), 518
- [CONTINUE](#) ([robot.running.model.Return](#) attribute), 528
- [CONTINUE](#) ([robot.running.model.Try](#) attribute), 527
- [CONTINUE](#) ([robot.running.model.TryBranch](#) attribute), 526
- [CONTINUE](#) ([robot.running.model.While](#) attribute), 521
- [continue\\_class](#) ([robot.model.body.BaseBody](#) attribute), 194
- [continue\\_class](#) ([robot.model.body.Body](#) attribute), 196
- [continue\\_class](#) ([robot.model.body.Branches](#) attribute), 198
- [continue\\_class](#) ([robot.result.model.Body](#) attribute), 422
- [continue\\_class](#) ([robot.result.model.Branches](#) attribute), 423
- [continue\\_class](#) ([robot.result.model.Iterations](#) attribute), 425
- [continue\\_class](#) ([robot.running.model.Body](#) attribute), 516
- [continue\\_for\\_loop\(\)](#)
  - ([robot.libraries.BuiltIn.BuiltIn](#) method), 42
- [continue\\_for\\_loop\\_if\(\)](#)
  - ([robot.libraries.BuiltIn.BuiltIn](#) method), 42
- [continue\\_on\\_failure](#) ([robot.errors.BreakLoop](#) at-

- [tribute\), 579](#)
- [continue\\_on\\_failure \(robot.errors.ContinueLoop attribute\), 578](#)
- [continue\\_on\\_failure \(robot.errors.ExecutionFailed attribute\), 576](#)
- [continue\\_on\\_failure \(robot.errors.ExecutionFailures attribute\), 577](#)
- [continue\\_on\\_failure \(robot.errors.ExecutionPassed attribute\), 578](#)
- [continue\\_on\\_failure \(robot.errors.ExecutionStatus attribute\), 576](#)
- [continue\\_on\\_failure \(robot.errors.HandlerExecutionFailed attribute\), 577](#)
- [continue\\_on\\_failure \(robot.errors.PassExecution attribute\), 578](#)
- [continue\\_on\\_failure \(robot.errors.ReturnFromKeyword attribute\), 579](#)
- [continue\\_on\\_failure \(robot.errors.UserKeywordExecutionFailed attribute\), 577](#)
- [ContinueHandler \(class in robot.result.xmlélémenthandlers\), 486](#)
- [ContinueLexer \(class in robot.parsing.lexer.statementlexers\), 298](#)
- [ContinueLoop, 578](#)
- [convert \(\) \(robot.running.arguments.argumentconverter.ArgumentConverter method\), 492](#)
- [convert \(\) \(robot.running.arguments.argumentspec.Argumentspec \(robot.libraries.BuiltIn.BuiltIn method\), 494](#)
- [convert \(\) \(robot.running.arguments.typeconverters.BooleanConverter method\), 497](#)
- [convert \(\) \(robot.running.arguments.typeconverters.ByteArrayConverter method\), 499](#)
- [convert \(\) \(robot.running.arguments.typeconverters.BytesConverter method\), 498](#)
- [convert \(\) \(robot.running.arguments.typeconverters.CombinedConverter method\), 503](#)
- [convert \(\) \(robot.running.arguments.typeconverters.CustomConverter method\), 503](#)
- [convert \(\) \(robot.running.arguments.typeconverters.DateConverter method\), 499](#)
- [convert \(\) \(robot.running.arguments.typeconverters.DateTimeConverter method\), 499](#)
- [convert \(\) \(robot.running.arguments.typeconverters.DecimalConverter method\), 498](#)
- [convert \(\) \(robot.running.arguments.typeconverters.DictionaryConverter method\), 501](#)
- [convert \(\) \(robot.running.arguments.typeconverters.EnumConverter method\), 496](#)
- [convert \(\) \(robot.running.arguments.typeconverters.FloatConverter method\), 497](#)
- [convert \(\) \(robot.running.arguments.typeconverters.FrozenSetConverter method\), 502](#)
- [convert \(\) \(robot.running.arguments.typeconverters.IntegerConverter method\), 497](#)
- [convert \(\) \(robot.running.arguments.typeconverters.ListConverter method\), 501](#)
- [convert \(\) \(robot.running.arguments.typeconverters.NoneConverter method\), 500](#)
- [convert \(\) \(robot.running.arguments.typeconverters.SetConverter method\), 502](#)
- [convert \(\) \(robot.running.arguments.typeconverters.StringConverter method\), 496](#)
- [convert \(\) \(robot.running.arguments.typeconverters.TimeDeltaConverter method\), 500](#)
- [convert \(\) \(robot.running.arguments.typeconverters.TupleConverter method\), 501](#)
- [convert \(\) \(robot.running.arguments.typeconverters.TypeConverter method\), 495](#)
- [convert \(\) \(robot.testdoc.JsonConverter method\), 585](#)
- [convert\\_date \(\) \(in robot.libraries.DateTime\), 75](#)
- [convert\\_docs\\_to\\_html \(\) \(robot.libdocpkg.model.LibraryDoc method\), 38](#)
- [convert\\_time \(\) \(in robot.libraries.DateTime\), 75](#)
- [convert\\_to\\_binary \(\) \(robot.libraries.BuiltIn.BuiltIn method\), 43](#)
- [convert\\_to\\_boolean \(\) \(robot.libraries.BuiltIn.BuiltIn method\), 43](#)
- [convert\\_to\\_bytes \(\) \(robot.libraries.BuiltIn.BuiltIn method\), 43](#)
- [convert\\_to\\_dictionary \(\) \(robot.libraries.Collections.Collections method\), 67](#)
- [convert\\_to\\_hex \(\) \(robot.libraries.BuiltIn.BuiltIn method\), 43](#)
- [convert\\_to\\_integer \(\) \(robot.libraries.BuiltIn.BuiltIn method\), 44](#)
- [convert\\_to\\_list \(\) \(robot.libraries.Collections.Collections method\), 67](#)
- [convert\\_to\\_lower\\_case \(\) \(robot.libraries.String.String method\), 97](#)
- [convert\\_to\\_number \(\) \(robot.libraries.BuiltIn.BuiltIn method\), 44](#)



<code>convert_to_octal()</code>	<code>(robot.libraries.BuiltIn.BuiltIn method),</code>	<code>robot.running.arguments.customconverters),</code>
44		495
<code>convert_to_string()</code>	<code>(robot.libraries.BuiltIn.BuiltIn method),</code>	<code>copy () (robot.model.body.BodyItem method),</code>
44		194
<code>convert_to_title_case()</code>	<code>(robot.libraries.String.String method),</code>	<code>copy () (robot.model.control.Break method),</code>
97		214
<code>convert_to_upper_case()</code>	<code>(robot.libraries.String.String method),</code>	<code>copy () (robot.model.control.Continue method),</code>
97		213
<code>convert_type_list_to_dict()</code>	<code>(robot.running.arguments.typevalidator.TypeValidator method),</code>	<code>copy () (robot.model.control.For method),</code>
503		204
<code>converter_for () (robot.running.arguments.typeconverter.BooleanConverter class method),</code>		<code>copy () (robot.model.control.If method),</code>
497		208
<code>converter_for () (robot.running.arguments.typeconverter.BytesConverter class method),</code>		<code>copy () (robot.model.control.IfBranch method),</code>
499		207
<code>converter_for () (robot.running.arguments.typeconverter.BytesArrayConverter class method),</code>		<code>copy () (robot.model.control.Return method),</code>
498		212
<code>converter_for () (robot.running.arguments.typeconverter.CombinedConverter class method),</code>		<code>copy () (robot.model.control.Try method),</code>
503		211
<code>converter_for () (robot.running.arguments.typeconverter.CustomConverter class method),</code>		<code>copy () (robot.model.control.TryBranch method),</code>
503		210
<code>converter_for () (robot.running.arguments.typeconverter.DateConverter class method),</code>		<code>copy () (robot.model.control.While method),</code>
499		206
<code>converter_for () (robot.running.arguments.typeconverter.DateTimeConverter class method),</code>		<code>copy () (robot.model.keyword.Keyword method),</code>
499		226
<code>converter_for () (robot.running.arguments.typeconverter.DecimalConverter class method),</code>		<code>robot.model.message.Message method),</code>
498		228
<code>converter_for () (robot.running.arguments.typeconverter.DictIndexConverter class method),</code>		<code>copy () (robot.model.metadata.Metadata method),</code>
501		229
<code>converter_for () (robot.running.arguments.typeconverter.DictIndexConverter class method),</code>		<code>robot.model.modelobject.ModelObject method),</code>
501		230
<code>converter_for () (robot.running.arguments.typeconverter.DictIndexConverter class method),</code>		<code>robot.model.testcase.TestCase method),</code>
501		249
<code>converter_for () (robot.running.arguments.typeconverter.DictIndexConverter class method),</code>		<code>copy () (robot.model.testsuite.TestSuite method),</code>
501		252
<code>converter_for () (robot.running.arguments.typeconverter.DictIndexConverter class method),</code>		<code>robot.model.utility.loggerhelper.Message method),</code>
501		275
<code>converter_for () (robot.running.arguments.typeconverter.DictIndexConverter class method),</code>		<code>robot.result.model.Break method),</code>
501		453
<code>converter_for () (robot.running.arguments.typeconverter.DictIndexConverter class method),</code>		<code>copy () (robot.result.model.Continue method),</code>
501		450
<code>converter_for () (robot.running.arguments.typeconverter.DictIndexConverter class method),</code>		<code>copy () (robot.result.model.For method),</code>
501		432
<code>converter_for () (robot.running.arguments.typeconverter.DictIndexConverter class method),</code>		<code>copy () (robot.result.model.ForIteration method),</code>
501		430
<code>converter_for () (robot.running.arguments.typeconverter.DictIndexConverter class method),</code>		<code>copy () (robot.result.model.If method),</code>
501		441
<code>converter_for () (robot.running.arguments.typeconverter.DictIndexConverter class method),</code>		<code>copy () (robot.result.model.IfBranch method),</code>
501		439
<code>converter_for () (robot.running.arguments.typeconverter.DictIndexConverter class method),</code>		<code>copy () (robot.result.model.Keyword method),</code>
501		455
<code>converter_for () (robot.running.arguments.typeconverter.DictIndexConverter class method),</code>		<code>copy () (robot.result.model.Message method),</code>
501		427
<code>converter_for () (robot.running.arguments.typeconverter.DictIndexConverter class method),</code>		<code>copy () (robot.result.model.Return method),</code>
501		448
<code>converter_for () (robot.running.arguments.typeconverter.DictIndexConverter class method),</code>		<code>copy () (robot.result.model.TestCase method),</code>
501		458
<code>converter_for () (robot.running.arguments.typeconverter.DictIndexConverter class method),</code>		<code>copy () (robot.result.model.TestSuite method),</code>
501		460
<code>converter_for () (robot.running.arguments.typeconverter.DictIndexConverter class method),</code>		<code>copy () (robot.result.model.Try method),</code>
501		446
<code>converter_for () (robot.running.arguments.typeconverter.DictIndexConverter class method),</code>		<code>copy () (robot.result.model.TryBranch method),</code>
501		443
<code>converter_for () (robot.running.arguments.typeconverter.DictIndexConverter class method),</code>		<code>copy () (robot.result.model.While method),</code>
501		437
<code>converter_for () (robot.running.arguments.typeconverter.DictIndexConverter class method),</code>		<code>copy () (robot.running.model.Break method),</code>
501		532
<code>converter_for () (robot.running.arguments.typeconverter.DictIndexConverter class method),</code>		<code>copy () (robot.running.model.Continue method),</code>
501		530
<code>converter_for () (robot.running.arguments.typeconverter.DictIndexConverter class method),</code>		<code>copy () (robot.running.model.For method),</code>
501		521
<code>converter_for () (robot.running.arguments.typeconverter.DictIndexConverter class method),</code>		<code>copy () (robot.running.model.If method),</code>
501		525
<code>converter_for () (robot.running.arguments.typeconverter.DictIndexConverter class method),</code>		<code>copy () (robot.running.model.IfBranch method),</code>
501		524
<code>converter_for () (robot.running.arguments.typeconverter.DictIndexConverter class method),</code>		<code>copy () (robot.running.model.Keyword method),</code>
501		518
<code>converter_for () (robot.running.arguments.typeconverter.DictIndexConverter class method),</code>		<code>copy () (robot.running.model.Return method),</code>
501		529
<code>converter_for () (robot.running.arguments.typeconverter.DictIndexConverter class method),</code>		<code>copy () (robot.running.model.TestCase method),</code>
501		533
<code>converter_for () (robot.running.arguments.typeconverter.DictIndexConverter class method),</code>		<code>copy () (robot.running.model.TestSuite method),</code>
501		536
<code>converter_for () (robot.running.arguments.typeconverter.DictIndexConverter class method),</code>		<code>copy () (robot.running.model.Try method),</code>
501		528
<code>converter_for () (robot.running.arguments.typeconverter.DictIndexConverter class method),</code>		<code>copy () (robot.running.model.TryBranch method),</code>
501		526
<code>converter_for () (robot.running.arguments.typeconverter.DictIndexConverter class method),</code>		<code>copy () (robot.running.model.While method),</code>
501		522
<code>converter_for () (robot.running.arguments.typeconverter.DictIndexConverter class method),</code>		<code>robot.utils.dotdict.DotDict method),</code>
501		558
<code>converter_for () (robot.running.arguments.typeconverter.DictIndexConverter class method),</code>		<code>robot.utils.normalizing.NormalizedDict method),</code>
501		566
<code>converter_for () (robot.running.arguments.typeconverter.DictIndexConverter class method),</code>		<code>robot.variables.scopes.GlobalVariables method),</code>
501		571
<code>ConverterInfo</code>	<code>(class in</code>	

`copy()` (*robot.variables.variables.Variables* method), 574  
`copy_dictionary()` (*robot.libraries.Collections.Collections* method), 67  
`copy_directory()` (*robot.libraries.OperatingSystem.OperatingSystem* method), 84  
`copy_element()` (*robot.libraries.XML.XML* method), 121  
`copy_file()` (*robot.libraries.OperatingSystem.OperatingSystem* method), 83  
`copy_files()` (*robot.libraries.OperatingSystem.OperatingSystem* method), 83  
`copy_list()` (*robot.libraries.Collections.Collections* method), 67  
`count()` (*robot.model.body.BaseBody* method), 195  
`count()` (*robot.model.body.Body* method), 196  
`count()` (*robot.model.body.Branches* method), 198  
`count()` (*robot.model.itemlist.ItemList* method), 224  
`count()` (*robot.model.keyword.Keywords* method), 227  
`count()` (*robot.model.message.Messages* method), 229  
`count()` (*robot.model.testcase.TestCases* method), 249  
`count()` (*robot.model.testsuite.TestSuites* method), 252  
`count()` (*robot.result.model.Body* method), 422  
`count()` (*robot.result.model.Branches* method), 423  
`count()` (*robot.result.model.Iterations* method), 425  
`count()` (*robot.running.model.Body* method), 516  
`count()` (*robot.running.model.Imports* method), 539  
`count_directories_in_directory()` (*robot.libraries.OperatingSystem.OperatingSystem* method), 87  
`count_files_in_directory()` (*robot.libraries.OperatingSystem.OperatingSystem* method), 87  
`count_items_in_directory()` (*robot.libraries.OperatingSystem.OperatingSystem* method), 87  
`count_values_in_list()` (*robot.libraries.Collections.Collections* method), 68  
`create` (*robot.model.body.BaseBody* attribute), 194  
`create` (*robot.model.body.Body* attribute), 196  
`create` (*robot.model.body.Branches* attribute), 198  
`create` (*robot.result.model.Body* attribute), 422  
`create` (*robot.result.model.Branches* attribute), 424  
`create` (*robot.result.model.Iterations* attribute), 425  
`create` (*robot.running.model.Body* attribute), 516  
`create()` (*robot.model.itemlist.ItemList* method), 224  
`create()` (*robot.model.keyword.Keywords* method), 226  
`create()` (*robot.model.message.Messages* method), 229  
`create()` (*robot.model.testcase.TestCases* method), 249  
`create()` (*robot.model.testsuite.TestSuites* method), 253  
`create()` (*robot.running.bodyrunner.DurationLimit* class method), 512  
`create()` (*robot.running.bodyrunner.InvalidLimit* class method), 512  
`create()` (*robot.running.bodyrunner.IterationCountLimit* class method), 512  
`create()` (*robot.running.bodyrunner.NoLimit* class method), 512  
`create()` (*robot.running.bodyrunner.WhileLimit* class method), 512  
`create()` (*robot.running.model.Imports* method), 539  
`create_binary_file()` (*robot.libraries.OperatingSystem.OperatingSystem* method), 82  
`create_branch()` (*robot.model.body.Branches* method), 197  
`create_branch()` (*robot.result.model.Branches* method), 424  
`create_break()` (*robot.model.body.BaseBody* method), 195  
`create_break()` (*robot.model.body.Body* method), 196  
`create_break()` (*robot.model.body.Branches* method), 198  
`create_break()` (*robot.result.model.Body* method), 422  
`create_break()` (*robot.result.model.Branches* method), 424  
`create_break()` (*robot.result.model.Iterations* method), 425  
`create_break()` (*robot.running.model.Body* method), 516  
`create_continue()` (*robot.model.body.BaseBody* method), 195  
`create_continue()` (*robot.model.body.Body* method), 196  
`create_continue()` (*robot.model.body.Branches* method), 198  
`create_continue()` (*robot.result.model.Body* method), 422  
`create_continue()` (*robot.result.model.Branches* method), 424  
`create_continue()` (*robot.result.model.Iterations* method), 425  
`create_continue()` (*robot.running.model.Body* method), 516  
`create_dictionary()` (*robot.libraries.BuiltIn.BuiltIn* method), 44  
`create_directory()` (*robot.libraries.OperatingSystem.OperatingSystem* method), 83

`create_file()` (*robot.libraries.OperatingSystem.OperatingSystem* method), 198  
`create_fixture()` (in module *robot.model.fixture*), 223  
`create_for()` (*robot.model.body.BaseBody* method), 194  
`create_for()` (*robot.model.body.Body* method), 196  
`create_for()` (*robot.model.body.Branches* method), 198  
`create_for()` (*robot.result.model.Body* method), 422  
`create_for()` (*robot.result.model.Branches* method), 424  
`create_for()` (*robot.result.model.Iterations* method), 425  
`create_for()` (*robot.running.model.Body* method), 516  
`create_if()` (*robot.model.body.BaseBody* method), 194  
`create_if()` (*robot.model.body.Body* method), 196  
`create_if()` (*robot.model.body.Branches* method), 198  
`create_if()` (*robot.result.model.Body* method), 422  
`create_if()` (*robot.result.model.Branches* method), 424  
`create_if()` (*robot.result.model.Iterations* method), 425  
`create_if()` (*robot.running.model.Body* method), 516  
`create_iteration()` (*robot.result.model.Iterations* method), 425  
`create_keyword()` (*robot.model.body.BaseBody* method), 194  
`create_keyword()` (*robot.model.body.Body* method), 196  
`create_keyword()` (*robot.model.body.Branches* method), 198  
`create_keyword()` (*robot.result.model.Body* method), 422  
`create_keyword()` (*robot.result.model.Branches* method), 424  
`create_keyword()` (*robot.result.model.Iterations* method), 425  
`create_keyword()` (*robot.running.model.Body* method), 516  
`create_link_target()` (*robot.reporting.jsbuildingcontext JsBuildingContext* method), 357  
`create_list()` (*robot.libraries.BuiltIn.BuiltIn* method), 45  
`create_message()` (*robot.model.body.BaseBody* method), 195  
`create_message()` (*robot.model.body.Body* method), 196  
`create_message()` (*robot.model.body.Branches* method), 198  
`create_message()` (*robot.result.model.Body* method), 422  
`create_message()` (*robot.result.model.Branches* method), 424  
`create_message()` (*robot.result.model.Iterations* method), 425  
`create_message()` (*robot.running.model.Body* method), 516  
`create_return()` (*robot.model.body.BaseBody* method), 195  
`create_return()` (*robot.model.body.Body* method), 196  
`create_return()` (*robot.model.body.Branches* method), 198  
`create_return()` (*robot.result.model.Body* method), 422  
`create_return()` (*robot.result.model.Branches* method), 424  
`create_return()` (*robot.result.model.Iterations* method), 425  
`create_return()` (*robot.running.model.Body* method), 516  
`create_runner()` (*robot.running.handlers.EmbeddedArgumentsHandler* method), 514  
`create_runner()` (*robot.running.handlerstore.HandlerStore* method), 514  
`create_runner()` (*robot.running.usererrorhandler.UserErrorHandler* method), 552  
`create_runner()` (*robot.running.userkeyword.EmbeddedArgumentsHandler* method), 552  
`create_runner()` (*robot.running.userkeyword.UserKeywordHandler* method), 552  
`create_setup()` (*robot.model.keyword.Keywords* method), 226  
`create_teardown()` (*robot.model.keyword.Keywords* method), 226  
`create_try()` (*robot.model.body.BaseBody* method), 194  
`create_try()` (*robot.model.body.Body* method), 196  
`create_try()` (*robot.model.body.Branches* method), 198  
`create_try()` (*robot.result.model.Body* method), 422  
`create_try()` (*robot.result.model.Branches* method), 424  
`create_try()` (*robot.result.model.Iterations* method), 425  
`create_try()` (*robot.running.model.Body* method), 516  
`create_while()` (*robot.model.body.BaseBody* method), 194  
`create_while()` (*robot.model.body.Body* method), 196

`create_while()` (*robot.model.body.Branches* method), 198

`create_while()` (*robot.result.model.Body* method), 422

`create_while()` (*robot.result.model.Branches* method), 424

`create_while()` (*robot.result.model.Iterations* method), 425

`create_while()` (*robot.running.model.Body* method), 516

`createLock()` (*robot.output.pyloggingconf.RobotHandler* method), 277

`critical` (*robot.result.model.TestCase* attribute), 457

`CssFileWriter` (class in *robot.htmldata.htmlfilewriter*), 34

`current` (*robot.model.sitestatistics.SuiteStatisticsBuilder* attribute), 241

`current` (*robot.running.context.ExecutionContexts* attribute), 513

`current` (*robot.utils.connectioncache.ConnectionCache* attribute), 558

`current` (*robot.variables.scopes.VariableScopes* attribute), 570

`current_index` (*robot.utils.connectioncache.ConnectionCache* attribute), 558

`current_output` (*robot.libraries.Telnet.TerminalEmulator* attribute), 111

`CUSTOM` (*robot.libdocpkg.datatypes.TypeDoc* attribute), 36

`CustomArgumentConverters` (class in *robot.running.arguments.customconverters*), 495

`CustomConverter` (class in *robot.running.arguments.typeconverters*), 503

**D**

`data` (*robot.running.modelcombiner.ModelCombiner* attribute), 539

`data_tokens` (*robot.parsing.model.statements.Arguments* attribute), 334

`data_tokens` (*robot.parsing.model.statements.Break* attribute), 349

`data_tokens` (*robot.parsing.model.statements.Comment* attribute), 350

`data_tokens` (*robot.parsing.model.statements.Continue* attribute), 348

`data_tokens` (*robot.parsing.model.statements.DefaultTags* attribute), 321

`data_tokens` (*robot.parsing.model.statements.Documentation* attribute), 319

`data_tokens` (*robot.parsing.model.statements.DocumentationOrMetadata* attribute), 312

`data_tokens` (*robot.parsing.model.statements.ElseHeader* attribute), 341

`data_tokens` (*robot.parsing.model.statements.ElseIfHeader* attribute), 340

`data_tokens` (*robot.parsing.model.statements.EmptyLine* attribute), 352

`data_tokens` (*robot.parsing.model.statements.End* attribute), 345

`data_tokens` (*robot.parsing.model.statements.Error* attribute), 351

`data_tokens` (*robot.parsing.model.statements.ExceptHeader* attribute), 344

`data_tokens` (*robot.parsing.model.statements.FinallyHeader* attribute), 344

`data_tokens` (*robot.parsing.model.statements.Fixture* attribute), 314

`data_tokens` (*robot.parsing.model.statements.ForceTags* attribute), 320

`data_tokens` (*robot.parsing.model.statements.ForHeader* attribute), 337

`data_tokens` (*robot.parsing.model.statements.IfElseHeader* attribute), 338

`data_tokens` (*robot.parsing.model.statements.IfHeader* attribute), 339

`data_tokens` (*robot.parsing.model.statements.InlineIfHeader* attribute), 339

`data_tokens` (*robot.parsing.model.statements.KeywordCall* attribute), 335

`data_tokens` (*robot.parsing.model.statements.KeywordName* attribute), 329

`data_tokens` (*robot.parsing.model.statements.LibraryImport* attribute), 316

`data_tokens` (*robot.parsing.model.statements.LoopControl* attribute), 348

`data_tokens` (*robot.parsing.model.statements.Metadata* attribute), 320

`data_tokens` (*robot.parsing.model.statements.MultiValue* attribute), 313

`data_tokens` (*robot.parsing.model.statements.NoArgumentHeader* attribute), 342

`data_tokens` (*robot.parsing.model.statements.ResourceImport* attribute), 317

`data_tokens` (*robot.parsing.model.statements.Return* attribute), 334

`data_tokens` (*robot.parsing.model.statements.ReturnStatement* attribute), 347

`data_tokens` (*robot.parsing.model.statements.SectionHeader* attribute), 315

`data_tokens` (*robot.parsing.model.statements.Setup* attribute), 330

`data_tokens` (*robot.parsing.model.statements.SingleValue* attribute), 313

`data_tokens` (*robot.parsing.model.statements.Statement* attribute), 311



[data\\_tokens \(robot.parsing.model.statements.SuiteSetup attribute\), 322](#)  
[data\\_tokens \(robot.parsing.model.statements.SuiteTeardown attribute\), 323](#)  
[data\\_tokens \(robot.parsing.model.statements.Tags attribute\), 331](#)  
[data\\_tokens \(robot.parsing.model.statements.Teardown attribute\), 330](#)  
[data\\_tokens \(robot.parsing.model.statements.Template attribute\), 332](#)  
[data\\_tokens \(robot.parsing.model.statements.TemplateArguments attribute\), 336](#)  
[data\\_tokens \(robot.parsing.model.statements.TestCaseName attribute\), 328](#)  
[data\\_tokens \(robot.parsing.model.statements.TestSetup attribute\), 324](#)  
[data\\_tokens \(robot.parsing.model.statements.TestTeardown attribute\), 325](#)  
[data\\_tokens \(robot.parsing.model.statements.TestTemplate attribute\), 325](#)  
[data\\_tokens \(robot.parsing.model.statements.TestTimeout attribute\), 326](#)  
[data\\_tokens \(robot.parsing.model.statements.Timeout attribute\), 333](#)  
[data\\_tokens \(robot.parsing.model.statements.TryHeader attribute\), 343](#)  
[data\\_tokens \(robot.parsing.model.statements.Variable attribute\), 327](#)  
[data\\_tokens \(robot.parsing.model.statements.Variables attribute\), 318](#)  
[data\\_tokens \(robot.parsing.model.statements.WhileHeader attribute\), 346](#)  
[DataError, 575](#)  
[DateConverter \(class in robot.running.arguments.typeconverters\), 499](#)  
[DateTimeConverter \(class in robot.running.arguments.typeconverters\), 499](#)  
[debug \(\) \(in module robot.api.logger\), 15](#)  
[debug \(\) \(in module robot.output.librarylogger\), 270](#)  
[debug \(\) \(robot.output.filelogger.FileLogger method\), 270](#)  
[debug \(\) \(robot.output.logger.Logger method\), 274](#)  
[debug \(\) \(robot.output.loggerhelper.AbstractLogger method\), 274](#)  
[debug \(\) \(robot.output.output.Output method\), 276](#)  
[debug \(\) \(robot.utils.importer.NoLogger method\), 564](#)  
[debug\\_file \(robot.conf.settings.RobotSettings attribute\), 31](#)  
[DebugFile \(\) \(in module robot.output.debugfile\), 270](#)  
[DecimalConverter \(class in robot.running.arguments.typeconverters\), 497](#)  
[decode\\_bytes\\_to\\_string \(\) \(robot.libraries.String.String method\), 98](#)  
[deepcopy \(\) \(robot.model.body.BodyItem method\), 194](#)  
[deepcopy \(\) \(robot.model.control.Break method\), 215](#)  
[deepcopy \(\) \(robot.model.control.Continue method\), 213](#)  
[deepcopy \(\) \(robot.model.control.For method\), 204](#)  
[deepcopy \(\) \(robot.model.control.If method\), 208](#)  
[deepcopy \(\) \(robot.model.control.IfBranch method\), 207](#)  
[deepcopy \(\) \(robot.model.control.Return method\), 212](#)  
[deepcopy \(\) \(robot.model.control.Try method\), 211](#)  
[deepcopy \(\) \(robot.model.control.TryBranch method\), 210](#)  
[deepcopy \(\) \(robot.model.control.While method\), 206](#)  
[deepcopy \(\) \(robot.model.keyword.Keyword method\), 226](#)  
[deepcopy \(\) \(robot.model.message.Message method\), 228](#)  
[deepcopy \(\) \(robot.model.modelobject.ModelObject method\), 230](#)  
[deepcopy \(\) \(robot.model.testcase.TestCase method\), 249](#)  
[deepcopy \(\) \(robot.model.testsuite.TestSuite method\), 252](#)  
[deepcopy \(\) \(robot.output.loggerhelper.Message method\), 275](#)  
[deepcopy \(\) \(robot.result.model.Break method\), 453](#)  
[deepcopy \(\) \(robot.result.model.Continue method\), 450](#)  
[deepcopy \(\) \(robot.result.model.For method\), 432](#)  
[deepcopy \(\) \(robot.result.model.ForIteration method\), 439](#)  
[deepcopy \(\) \(robot.result.model.If method\), 441](#)  
[deepcopy \(\) \(robot.result.model.IfBranch method\), 439](#)  
[deepcopy \(\) \(robot.result.model.Keyword method\), 456](#)  
[deepcopy \(\) \(robot.result.model.Message method\), 427](#)  
[deepcopy \(\) \(robot.result.model.Return method\), 448](#)  
[deepcopy \(\) \(robot.result.model.TestCase method\), 458](#)  
[deepcopy \(\) \(robot.result.model.TestSuite method\), 460](#)  
[deepcopy \(\) \(robot.result.model.Try method\), 446](#)  
[deepcopy \(\) \(robot.result.model.TryBranch method\), 443](#)  
[deepcopy \(\) \(robot.result.model.While method\), 437](#)  
[deepcopy \(\) \(robot.result.model.WhileIteration method\), 434](#)  
[deepcopy \(\) \(robot.running.model.Break method\), 532](#)  
[deepcopy \(\) \(robot.running.model.Continue method\),](#)

- 530
- `deepcopy()` (*robot.running.model.For* method), 521
- `deepcopy()` (*robot.running.model.If* method), 525
- `deepcopy()` (*robot.running.model.IfBranch* method), 524
- `deepcopy()` (*robot.running.model.Keyword* method), 519
- `deepcopy()` (*robot.running.model.Return* method), 529
- `deepcopy()` (*robot.running.model.TestCase* method), 533
- `deepcopy()` (*robot.running.model.TestSuite* method), 536
- `deepcopy()` (*robot.running.model.Try* method), 528
- `deepcopy()` (*robot.running.model.TryBranch* method), 526
- `deepcopy()` (*robot.running.model.While* method), 522
- `default_repr` (*robot.running.arguments.argumentspec* attribute), 494
- `DEFAULT_TAGS` (*robot.parsing.lexer.tokens.END* attribute), 304
- `DEFAULT_TAGS` (*robot.parsing.lexer.tokens.EOS* attribute), 302
- `DEFAULT_TAGS` (*robot.parsing.lexer.tokens.Token* attribute), 300
- `DefaultLogger` (class in *robot.utils.application*), 553
- `DefaultTags` (class in *robot.parsing.model.statements*), 321
- `DefaultValue` (class in *robot.running.arguments.argumentmapper*), 492
- `deiconify()` (*robot.libraries.dialogs\_py.InputDialog* method), 139
- `deiconify()` (*robot.libraries.dialogs\_py.MessageDialog* method), 125
- `deiconify()` (*robot.libraries.dialogs\_py.MultipleSelectionDialog* method), 167
- `deiconify()` (*robot.libraries.dialogs\_py.PassFailDialog* method), 181
- `deiconify()` (*robot.libraries.dialogs\_py.SelectionDialog* method), 153
- `delayed_logging` (*robot.output.logger.Logger* attribute), 273
- `deletecommand()` (*robot.libraries.dialogs\_py.InputDialog* method), 139
- `deletecommand()` (*robot.libraries.dialogs\_py.MessageDialog* method), 125
- `deletecommand()` (*robot.libraries.dialogs\_py.MultipleSelectionDialog* method), 167
- `deletecommand()` (*robot.libraries.dialogs\_py.PassFailDialog* method), 181
- `deletecommand()` (*robot.libraries.dialogs\_py.SelectionDialog* method), 153
- `denominator` (*robot.reporting.stringcache.StringIndex* attribute), 365
- `deprecated` (*robot.libdocpkg.model.KeywordDoc* attribute), 38
- `deprecated()` (in *robot.result.model.deprecation* module), 463
- `DeprecatedAttributesMixin` (class in *robot.result.model.deprecation*), 463
- `deprecation_message` (*robot.model.keyword.Keywords* attribute), 226
- `destroy()` (*robot.libraries.dialogs\_py.InputDialog* method), 139
- `destroy()` (*robot.libraries.dialogs\_py.MessageDialog* method), 125
- `destroy()` (*robot.libraries.dialogs\_py.MultipleSelectionDialog* method), 167
- `destroy()` (*robot.libraries.dialogs\_py.PassFailDialog* method), 181
- `destroy()` (*robot.libraries.dialogs\_py.SelectionDialog* method), 153
- `DictDumper` (class in *robot.htmldata.jsonwriter*), 34
- `dictionaries_should_be_equal()` (*robot.libraries.Collections.Collections* method), 68
- `dictionary_should_contain_item()` (*robot.libraries.Collections.Collections* method), 68
- `dictionary_should_contain_key()` (*robot.libraries.Collections.Collections* method), 68
- `dictionary_should_contain_sub_dictionary()` (*robot.libraries.Collections.Collections* method), 68
- `dictionary_should_contain_value()` (*robot.libraries.Collections.Collections* method), 68
- `dictionary_should_not_contain_key()` (*robot.libraries.Collections.Collections* method), 68
- `dictionary_should_not_contain_value()` (*robot.libraries.Collections.Collections* method), 68
- `DictionaryConverter` (class in *robot.running.arguments.typeconverters*), 501
- `DictToKwargs` (class in *robot.running.arguments.argumentresolver*), 476
- `DictVariableTableValue` (class in *robot.variables.tablessetter*), 573
- `directory` (*robot.running.model.Import* attribute), 539
- `directory_should_be_empty()` (*robot.libraries.OperatingSystem.OperatingSystem* attribute), 365

method), 82

directory\_should\_exist() (robot.libraries.OperatingSystem.OperatingSystem method), 81

directory\_should\_not\_be\_empty() (robot.libraries.OperatingSystem.OperatingSystem method), 82

directory\_should\_not\_exist() (robot.libraries.OperatingSystem.OperatingSystem method), 81

disable\_library\_import\_logging() (robot.output.logger.Logger method), 273

disable\_message\_cache() (robot.output.logger.Logger method), 273

discard\_suite\_scope() (robot.output.listenermethods.LibraryListenerMethods method), 272

discard\_suite\_scope() (robot.output.listeners.LibraryListeners method), 272

doc (robot.libdocpkg.model.LibraryDoc attribute), 37

doc (robot.model.keyword.Keyword attribute), 224

doc (robot.model.stats.TagStat attribute), 241

doc (robot.model.testcase.TestCase attribute), 248

doc (robot.model.testsuite.TestSuite attribute), 250

doc (robot.result.model.Break attribute), 452

doc (robot.result.model.Continue attribute), 449

doc (robot.result.model.For attribute), 431

doc (robot.result.model.ForIteration attribute), 429

doc (robot.result.model.If attribute), 440

doc (robot.result.model.IfBranch attribute), 438

doc (robot.result.model.Keyword attribute), 456

doc (robot.result.model.Return attribute), 447

doc (robot.result.model.TestCase attribute), 458

doc (robot.result.model.TestSuite attribute), 460

doc (robot.result.model.Try attribute), 445

doc (robot.result.model.TryBranch attribute), 442

doc (robot.result.model.While attribute), 436

doc (robot.result.model.WhileIteration attribute), 433

doc (robot.running.arguments.customconverters.ConverterInfo attribute), 495

doc (robot.running.arguments.typeconverters.BooleanConverter attribute), 497

doc (robot.running.arguments.typeconverters.ByteArrayConverter attribute), 499

doc (robot.running.arguments.typeconverters.BytesConverter attribute), 498

doc (robot.running.arguments.typeconverters.CombinedConverter attribute), 503

doc (robot.running.arguments.typeconverters.CustomConverter attribute), 503

doc (robot.running.arguments.typeconverters.DateConverter attribute), 499

doc (robot.running.arguments.typeconverters.DateTimeConverter attribute), 578

doc (robot.running.arguments.typeconverters.DecimalConverter attribute), 498

doc (robot.running.arguments.typeconverters.DictionaryConverter attribute), 501

doc (robot.running.arguments.typeconverters.EnumConverter attribute), 496

doc (robot.running.arguments.typeconverters.FloatConverter attribute), 497

doc (robot.running.arguments.typeconverters.FrozenSetConverter attribute), 502

doc (robot.running.arguments.typeconverters.IntegerConverter attribute), 497

doc (robot.running.arguments.typeconverters.ListConverter attribute), 501

doc (robot.running.arguments.typeconverters.NoneConverter attribute), 500

doc (robot.running.arguments.typeconverters.SetConverter attribute), 502

doc (robot.running.arguments.typeconverters.StringConverter attribute), 496

doc (robot.running.arguments.typeconverters.TimeDeltaConverter attribute), 500

doc (robot.running.arguments.typeconverters.TupleConverter attribute), 501

doc (robot.running.arguments.typeconverters.TypeConverter attribute), 495

doc (robot.running.model.Keyword attribute), 519

doc (robot.running.model.TestCase attribute), 533

doc (robot.running.model.TestSuite attribute), 536

doc (robot.running.usererrorhandler.UserErrorHandler attribute), 551

doc\_format (robot.libdocpkg.model.LibraryDoc attribute), 37

DocFormatter (class in robot.libdocpkg.htmlutils), 36

DocHandler (class in robot.result.xmllelementhandlers), 487

DocToHtml (class in robot.libdocpkg.htmlutils), 36

Documentation (class in robot.parsing.model.statements), 318

DOCUMENTATION (robot.parsing.lexer.tokens.END attribute), 304

DOCUMENTATION (robot.parsing.lexer.tokens.EOS attribute), 302

DOCUMENTATION (robot.parsing.lexer.tokens.Token attribute), 299

DocumentationBuilder() (in module robot.libdocpkg.builder), 35

DocumentationOrMetadata (class in robot.parsing.model.statements), 312

dont\_continue (robot.errors.BreakLoop attribute), 579

dont\_continue (robot.errors.ContinueLoop attribute), 578

- `dont_continue` (*robot.errors.ExecutionFailed* attribute), 576
- `dont_continue` (*robot.errors.ExecutionFailures* attribute), 577
- `dont_continue` (*robot.errors.ExecutionPassed* attribute), 578
- `dont_continue` (*robot.errors.ExecutionStatus* attribute), 576
- `dont_continue` (*robot.errors.HandlerExecutionFailed* attribute), 577
- `dont_continue` (*robot.errors.PassExecution* attribute), 578
- `dont_continue` (*robot.errors.ReturnFromKeyword* attribute), 579
- `dont_continue` (*robot.errors.UserKeywordExecutionFailed* attribute), 577
- `DosHighlighter` (class in *robot.output.console.highlighting*), 268
- `DotDict` (class in *robot.utils.dotdict*), 558
- `DottedImporter` (class in *robot.utils.importer*), 563
- `DottedOutput` (class in *robot.output.console.dotted*), 263
- `dry_run` (*robot.conf.settings.RobotSettings* attribute), 31
- `dry_run()` (*robot.running.librarykeywordrunner.EmbeddedArgumentParser* method), 515
- `dry_run()` (*robot.running.librarykeywordrunner.LibraryKeywordRunner* method), 515
- `dry_run()` (*robot.running.librarykeywordrunner.RunKeywordRunner* method), 515
- `dry_run()` (*robot.running.usererrorhandler.UserErrorHandler* method), 552
- `dry_run()` (*robot.running.userkeywordrunner.EmbeddedArgumentsRunner* method), 552
- `dry_run()` (*robot.running.userkeywordrunner.UserKeywordRunner* method), 552
- `dump()` (*robot.htmldata.jsonwriter.DictDumper* method), 34
- `dump()` (*robot.htmldata.jsonwriter.IntegerDumper* method), 34
- `dump()` (*robot.htmldata.jsonwriter.JsonDumper* method), 34
- `dump()` (*robot.htmldata.jsonwriter.MappingDumper* method), 35
- `dump()` (*robot.htmldata.jsonwriter.NoneDumper* method), 35
- `dump()` (*robot.htmldata.jsonwriter.StringDumper* method), 34
- `dump()` (*robot.htmldata.jsonwriter.TupleListDumper* method), 35
- `dump()` (*robot.reporting.stringcache.StringCache* method), 366
- `DurationLimit` (class in *robot.running.bodyrunner*), 512
- `DynamicArgumentParser` (class in *robot.running.arguments.argumentparser*), 493
- `DynamicHandler()` (in module *robot.running.handlers*), 513
- ## E
- `earlier_failures` (*robot.errors.BreakLoop* attribute), 579
- `earlier_failures` (*robot.errors.ContinueLoop* attribute), 578
- `earlier_failures` (*robot.errors.ExecutionPassed* attribute), 578
- `earlier_failures` (*robot.errors.PassExecution* attribute), 578
- `earlier_failures` (*robot.errors.ReturnFromKeyword* attribute), 579
- `elapsed` (*robot.model.stats.Stat* attribute), 240
- `elapsed` (*robot.model.stats.SuiteStat* attribute), 240
- `elapsedtime` (*robot.result.model.Break* attribute), 453
- `elapsedtime` (*robot.result.model.Continue* attribute), 451
- `elapsedtime` (*robot.result.model.For* attribute), 432
- `elapsedtime` (*robot.result.model.ForIteration* attribute), 430
- `elapsedtime` (*robot.result.model.If* attribute), 441
- `elapsedtime` (*robot.result.model.IfBranch* attribute), 439
- `elapsedtime` (*robot.result.model.Keyword* attribute), 456
- `elapsedtime` (*robot.result.model.Return* attribute), 448
- `elapsedtime` (*robot.result.model.StatusMixin* attribute), 428
- `elapsedtime` (*robot.result.model.TestCase* attribute), 458
- `elapsedtime` (*robot.result.model.TestSuite* attribute), 463
- `elapsedtime` (*robot.result.model.Try* attribute), 446
- `elapsedtime` (*robot.result.model.TryBranch* attribute), 444
- `elapsedtime` (*robot.result.model.While* attribute), 437
- `elapsedtime` (*robot.result.model.WhileIteration* attribute), 435
- `element()` (*robot.utils.markupwriters.HtmlWriter* method), 564
- `element()` (*robot.utils.markupwriters.NullMarkupWriter* method), 565
- `element()` (*robot.utils.markupwriters.XmlWriter* method), 564
- `element_attribute_should_be()` (*robot.libraries.XML.XML* method), 118
- `element_attribute_should_match()` (*robot.libraries.XML.XML* method), 118



element\_handlers (robot.result.xml\_element\_handlers.ArgumentHandler attribute), 490  
 element\_handlers (robot.result.xml\_element\_handlers.TestHandler attribute), 484  
 element\_handlers (robot.result.xml\_element\_handlers.TimeoutHandler attribute), 489  
 element\_handlers (robot.result.xml\_element\_handlers.TryHandler attribute), 485  
 element\_handlers (robot.result.xml\_element\_handlers.ValueHandler attribute), 490  
 element\_handlers (robot.result.xml\_element\_handlers.VarHandler attribute), 489  
 element\_handlers (robot.result.xml\_element\_handlers.WhileHandler attribute), 484  
 element\_handlers (robot.result.xml\_element\_handlers.\_should\_exist() attribute), 487  
 element\_handlers (robot.result.xml\_element\_handlers.\_should\_not\_exist() attribute), 483  
 element\_handlers (robot.result.xml\_element\_handlers.\_not\_have\_attribute() attribute), 490  
 element\_handlers (robot.result.xml\_element\_handlers.\_text\_should\_be() attribute), 490  
 element\_handlers (robot.result.xml\_element\_handlers.\_text\_should\_match() attribute), 484  
 element\_handlers (robot.result.xml\_element\_handlers.\_to\_string() attribute), 485  
 element\_handlers (robot.result.xml\_element\_handlers.\_comparator (class in robot.libraries.XML), attribute), 485  
 element\_handlers (robot.result.xml\_element\_handlers.\_KeywordHandler (class in robot.libraries.XML), attribute), 484  
 element\_handlers (robot.result.xml\_element\_handlers.\_in attribute), 483  
 element\_handlers (robot.result.xml\_element\_handlers.\_elements\_should\_be\_equal() attribute), 487  
 element\_handlers (robot.result.xml\_element\_handlers.\_method (robot.libraries.XML.XML method), attribute), 487  
 element\_handlers (robot.result.xml\_element\_handlers.\_method (robot.libraries.XML.XML method), attribute), 488  
 element\_handlers (robot.result.xml\_element\_handlers.\_method (robot.model.body.BodyItem attribute), attribute), 488  
 element\_handlers (robot.result.xml\_element\_handlers.\_method (robot.model.control.Break attribute), attribute), 486  
 element\_handlers (robot.result.xml\_element\_handlers.\_method (robot.model.control.Continue attribute), attribute), 486  
 element\_handlers (robot.result.xml\_element\_handlers.\_method (robot.model.control.For attribute), attribute), 486  
 element\_handlers (robot.result.xml\_element\_handlers.\_method (robot.model.control.If attribute), attribute), 486  
 element\_handlers (robot.result.xml\_element\_handlers.\_method (robot.model.control.IfBranch attribute), attribute), 486  
 element\_handlers (robot.result.xml\_element\_handlers.\_method (robot.model.control.Return attribute), attribute), 483  
 element\_handlers (robot.result.xml\_element\_handlers.\_method (robot.model.control.Try attribute), attribute), 483  
 element\_handlers (robot.result.xml\_element\_handlers.\_method (robot.model.control.TryBranch attribute), attribute), 483  
 element\_handlers (robot.result.xml\_element\_handlers.\_method (robot.model.control.While attribute), attribute), 483  
 element\_handlers (robot.result.xml\_element\_handlers.\_method (robot.model.message.Message attribute), attribute), 491  
 element\_handlers (robot.result.xml\_element\_handlers.\_method (robot.output.loggerhelper.Message attribute), attribute), 487  
 element\_handlers (robot.result.xml\_element\_handlers.\_method (robot.parsing.lexer.tokens.END attribute), attribute), 487  
 element\_handlers (robot.result.xml\_element\_handlers.\_method (robot.parsing.lexer.tokens.EOS attribute), attribute), 484  
 element\_handlers (robot.result.xml\_element\_handlers.\_method (robot.parsing.lexer.tokens.Token attribute), attribute), 484  
 element\_handlers (robot.result.xml\_element\_handlers.\_method (robot.result.model.Continue attribute), attribute), 488  
 element\_handlers (robot.result.xml\_element\_handlers.\_method (robot.result.model.For attribute), attribute), 488  
 element\_handlers (robot.result.xml\_element\_handlers.\_method (robot.result.model.ForIteration attribute), attribute), 488  
 element\_handlers (robot.result.xml\_element\_handlers.\_method (robot.result.model.If attribute), attribute), 488

- ELSE (*robot.result.model.IfBranch* attribute), 438
- ELSE (*robot.result.model.Keyword* attribute), 455
- ELSE (*robot.result.model.Message* attribute), 427
- ELSE (*robot.result.model.Return* attribute), 447
- ELSE (*robot.result.model.Try* attribute), 445
- ELSE (*robot.result.model.TryBranch* attribute), 442
- ELSE (*robot.result.model.While* attribute), 436
- ELSE (*robot.result.model.WhileIteration* attribute), 434
- ELSE (*robot.running.model.Break* attribute), 531
- ELSE (*robot.running.model.Continue* attribute), 530
- ELSE (*robot.running.model.For* attribute), 520
- ELSE (*robot.running.model.If* attribute), 524
- ELSE (*robot.running.model.IfBranch* attribute), 523
- ELSE (*robot.running.model.Keyword* attribute), 518
- ELSE (*robot.running.model.Return* attribute), 528
- ELSE (*robot.running.model.Try* attribute), 527
- ELSE (*robot.running.model.TryBranch* attribute), 526
- ELSE (*robot.running.model.While* attribute), 522
- else\_branch (*robot.model.control.Try* attribute), 210
- else\_branch (*robot.result.model.Try* attribute), 446
- else\_branch (*robot.running.model.Try* attribute), 528
- ELSE\_IF (*robot.model.body.BodyItem* attribute), 193
- ELSE\_IF (*robot.model.control.Break* attribute), 214
- ELSE\_IF (*robot.model.control.Continue* attribute), 213
- ELSE\_IF (*robot.model.control.For* attribute), 204
- ELSE\_IF (*robot.model.control.If* attribute), 208
- ELSE\_IF (*robot.model.control.IfBranch* attribute), 206
- ELSE\_IF (*robot.model.control.Return* attribute), 212
- ELSE\_IF (*robot.model.control.Try* attribute), 210
- ELSE\_IF (*robot.model.control.TryBranch* attribute), 209
- ELSE\_IF (*robot.model.control.While* attribute), 205
- ELSE\_IF (*robot.model.keyword.Keyword* attribute), 225
- ELSE\_IF (*robot.model.message.Message* attribute), 228
- ELSE\_IF (*robot.output.loggerhelper.Message* attribute), 274
- ELSE\_IF (*robot.parsing.lexer.tokens.END* attribute), 304
- ELSE\_IF (*robot.parsing.lexer.tokens.EOS* attribute), 302
- ELSE\_IF (*robot.parsing.lexer.tokens.Token* attribute), 300
- ELSE\_IF (*robot.result.model.Break* attribute), 452
- ELSE\_IF (*robot.result.model.Continue* attribute), 450
- ELSE\_IF (*robot.result.model.For* attribute), 431
- ELSE\_IF (*robot.result.model.ForIteration* attribute), 429
- ELSE\_IF (*robot.result.model.If* attribute), 440
- ELSE\_IF (*robot.result.model.IfBranch* attribute), 438
- ELSE\_IF (*robot.result.model.Keyword* attribute), 455
- ELSE\_IF (*robot.result.model.Message* attribute), 427
- ELSE\_IF (*robot.result.model.Return* attribute), 447
- ELSE\_IF (*robot.result.model.Try* attribute), 445
- ELSE\_IF (*robot.result.model.TryBranch* attribute), 443
- ELSE\_IF (*robot.result.model.While* attribute), 436
- ELSE\_IF (*robot.result.model.WhileIteration* attribute), 434
- ELSE\_IF (*robot.running.model.Break* attribute), 531
- ELSE\_IF (*robot.running.model.Continue* attribute), 530
- ELSE\_IF (*robot.running.model.For* attribute), 520
- ELSE\_IF (*robot.running.model.If* attribute), 524
- ELSE\_IF (*robot.running.model.IfBranch* attribute), 523
- ELSE\_IF (*robot.running.model.Keyword* attribute), 518
- ELSE\_IF (*robot.running.model.Return* attribute), 528
- ELSE\_IF (*robot.running.model.Try* attribute), 527
- ELSE\_IF (*robot.running.model.TryBranch* attribute), 526
- ELSE\_IF (*robot.running.model.While* attribute), 522
- ElseHeader (class in *robot.parsing.model.statements*), 341
- ElseHeaderLexer (class in *robot.parsing.lexer.statementslexers*), 297
- ElseIfHeader (class in *robot.parsing.model.statements*), 340
- ElseIfHeaderLexer (class in *robot.parsing.lexer.statementslexers*), 297
- EmbeddedArgumentParser (class in *robot.running.arguments.embedded*), 495
- EmbeddedArguments (class in *robot.running.arguments.embedded*), 495
- EmbeddedArgumentsHandler (class in *robot.running.handlers*), 514
- EmbeddedArgumentsHandler (class in *robot.running.userkeyword*), 552
- EmbeddedArgumentsRunner (class in *robot.running.librarykeywordrunner*), 515
- EmbeddedArgumentsRunner (class in *robot.running.userkeywordrunner*), 552
- emit () (*robot.output.pyloggingconf.RobotHandler* method), 277
- empty (*robot.variables.finders.EmptyFinder* attribute), 569
- empty\_cache () (*robot.utils.connectioncache.ConnectionCache* method), 558
- empty\_directory () (*robot.libraries.OperatingSystem.OperatingSystem* method), 83
- EmptyFinder (class in *robot.variables.finders*), 569
- EmptyLine (class in *robot.parsing.model.statements*), 352
- EmptySuiteRemover (class in *robot.model.filter*), 215
- enable\_library\_import\_logging () (*robot.output.logger.Logger* method), 273
- encode\_string\_to\_bytes () (*robot.libraries.String.String* method), 98
- encode\_threshold (*robot.libraries.Remote.TimeoutHTTPSTransport* attribute), 95

`encode_threshold` (*robot.libraries.Remote.TimeoutHTTPTransport* method), 487  
`attribute`), 94  
`END` (class in *robot.parsing.lexer.tokens*), 303  
`End` (class in *robot.parsing.model.statements*), 345  
`END` (*robot.parsing.lexer.tokens.END* attribute), 304  
`END` (*robot.parsing.lexer.tokens.EOS* attribute), 302  
`END` (*robot.parsing.lexer.tokens.Token* attribute), 300  
`end()` (*robot.result.xmlelementhandlers.ArgumentHandler* method), 489  
`end()` (*robot.result.xmlelementhandlers.ArgumentsHandler* method), 489  
`end()` (*robot.result.xmlelementhandlers.AssignHandler* method), 489  
`end()` (*robot.result.xmlelementhandlers.BranchHandler* method), 485  
`end()` (*robot.result.xmlelementhandlers.BreakHandler* method), 486  
`end()` (*robot.result.xmlelementhandlers.ContinueHandler* method), 486  
`end()` (*robot.result.xmlelementhandlers.DocHandler* method), 487  
`end()` (*robot.result.xmlelementhandlers.ElementHandler* method), 483  
`end()` (*robot.result.xmlelementhandlers.ErrorMessageHandler* method), 490  
`end()` (*robot.result.xmlelementhandlers.ErrorsHandler* method), 490  
`end()` (*robot.result.xmlelementhandlers.ForHandler* method), 484  
`end()` (*robot.result.xmlelementhandlers.IfHandler* method), 485  
`end()` (*robot.result.xmlelementhandlers.IterationHandler* method), 485  
`end()` (*robot.result.xmlelementhandlers.KeywordHandler* method), 484  
`end()` (*robot.result.xmlelementhandlers.MessageHandler* method), 487  
`end()` (*robot.result.xmlelementhandlers.MetadataHandler* method), 487  
`end()` (*robot.result.xmlelementhandlers.MetadataItemHandler* method), 488  
`end()` (*robot.result.xmlelementhandlers.MetaHandler* method), 488  
`end()` (*robot.result.xmlelementhandlers.PatternHandler* method), 486  
`end()` (*robot.result.xmlelementhandlers.ReturnHandler* method), 486  
`end()` (*robot.result.xmlelementhandlers.RobotHandler* method), 483  
`end()` (*robot.result.xmlelementhandlers.RootHandler* method), 483  
`end()` (*robot.result.xmlelementhandlers.StatisticsHandler* method), 491  
`end()` (*robot.result.xmlelementhandlers.StatusHandler* method), 484  
`end()` (*robot.result.xmlelementhandlers.SuiteHandler* method), 484  
`end()` (*robot.result.xmlelementhandlers.TagHandler* method), 488  
`end()` (*robot.result.xmlelementhandlers.TagsHandler* method), 488  
`end()` (*robot.result.xmlelementhandlers.TestHandler* method), 484  
`end()` (*robot.result.xmlelementhandlers.TimeoutHandler* method), 489  
`end()` (*robot.result.xmlelementhandlers.TryHandler* method), 485  
`end()` (*robot.result.xmlelementhandlers.ValueHandler* method), 490  
`end()` (*robot.result.xmlelementhandlers.VarHandler* method), 489  
`end()` (*robot.result.xmlelementhandlers.WhileHandler* method), 484  
`end()` (*robot.result.xmlelementhandlers.XmlElementHandler* method), 483  
`end()` (*robot.utils.htmlformatters.HeaderFormatter* method), 562  
`end()` (*robot.utils.htmlformatters.ListFormatter* method), 562  
`end()` (*robot.utils.htmlformatters.ParagraphFormatter* method), 562  
`end()` (*robot.utils.htmlformatters.PreformattedFormatter* method), 562  
`end()` (*robot.utils.htmlformatters.RulerFormatter* method), 561  
`end()` (*robot.utils.htmlformatters.TableFormatter* method), 562  
`end()` (*robot.utils.markupwriters.HtmlWriter* method), 564  
`end()` (*robot.utils.markupwriters.NullMarkupWriter* method), 565  
`end()` (*robot.utils.markupwriters.XmlWriter* method), 565  
`end_block()` (*robot.parsing.model.blocks.ValidationContext* method), 310  
`end_body_item()` (*robot.conf.gatherfailed.GatherFailedSuites* method), 27  
`end_body_item()` (*robot.conf.gatherfailed.GatherFailedTests* method), 23  
`end_body_item()` (*robot.model.configurer.SuiteConfigurer* method), 199  
`end_body_item()` (*robot.model.filter.EmptySuiteRemover* method), 215  
`end_body_item()` (*robot.model.filter.Filter* method), 219  
`end_body_item()` (*robot.model.modifier.ModelModifier* method), 230  
`end_body_item()` (*robot.model.statistics.StatisticsBuilder* method), 230

`method`), 235  
`end_body_item()` (`robot.model.tagsetter.TagSetter` `method`), 243  
`end_body_item()` (`robot.model.totalstatistics.TotalStatisticsBuilder` `method`), 254  
`end_body_item()` (`robot.model.visitor.SuiteVisitor` `method`), 263  
`end_body_item()` (`robot.output.console.dotted.StatusReporter` `method`), 264  
`end_body_item()` (`robot.output.xmllogger.XmlLogger` `method`), 281  
`end_body_item()` (`robot.reporting.outputwriter.OutputWriter` `method`), 360  
`end_body_item()` (`robot.reporting.xunitwriter.XUnitFileWriter` `method`), 366  
`end_body_item()` (`robot.result.configurer.SuiteConfigurer` `method`), 372  
`end_body_item()` (`robot.result.keywordremover.AllKeywordsRemover` `method`), 380  
`end_body_item()` (`robot.result.keywordremover.ByTagNameKeywordRemover` `method`), 388  
`end_body_item()` (`robot.result.keywordremover.ByTagKeywordRemover` `method`), 392  
`end_body_item()` (`robot.result.keywordremover.ForLoopItemsRemover` `method`), 396  
`end_body_item()` (`robot.result.keywordremover.PassedKeywordRemover` `method`), 384  
`end_body_item()` (`robot.result.keywordremover.WaitUntilKeywordSucceedsRemover` `method`), 404  
`end_body_item()` (`robot.result.keywordremover.WarningAndErrorFinder` `method`), 409  
`end_body_item()` (`robot.result.keywordremover.WhileLoopItemsRemover` `method`), 400  
`end_body_item()` (`robot.result.merger.Merger` `method`), 413  
`end_body_item()` (`robot.result.messagefilter.MessageFilter` `method`), 417  
`end_body_item()` (`robot.result.resultbuilder.RemoveKeywords` `method`), 465  
`end_body_item()` (`robot.result.suiteteardownfailed.SuiteTeardownFailed` `method`), 473  
`end_body_item()` (`robot.result.suiteteardownfailed.SuiteTeardownFailureHandler` `method`), 469  
`end_body_item()` (`robot.result.visitor.ResultVisitor` `method`), 478  
`end_body_item()` (`robot.running.randomizer.Randomizer` `method`), 541  
`end_body_item()` (`robot.running.suiterunner.SuiteRunner` `method`), 547  
`end_break()` (`robot.conf.gatherfailed.GatherFailedSuites` `method`), 27  
`end_break()` (`robot.conf.gatherfailed.GatherFailedTests` `method`), 23  
`end_break()` (`robot.model.configurer.SuiteConfigurer` `method`), 199  
`end_break()` (`robot.model.filter.EmptySuiteRemover` `method`), 215  
`end_break()` (`robot.model.filter.Filter` `method`), 220  
`end_break()` (`robot.model.modifier.ModelModifier` `method`), 231  
`end_break()` (`robot.model.statistics.StatisticsBuilder` `method`), 236  
`end_break()` (`robot.model.tagsetter.TagSetter` `method`), 243  
`end_break()` (`robot.model.totalstatistics.TotalStatisticsBuilder` `method`), 254  
`end_break()` (`robot.model.visitor.SuiteVisitor` `method`), 262  
`end_break()` (`robot.output.console.dotted.StatusReporter` `method`), 264  
`end_break()` (`robot.output.xmllogger.XmlLogger` `method`), 280  
`end_break()` (`robot.reporting.outputwriter.OutputWriter` `method`), 360  
`end_break()` (`robot.reporting.xunitwriter.XUnitFileWriter` `method`), 367  
`end_break()` (`robot.result.configurer.SuiteConfigurer` `method`), 372  
`end_break()` (`robot.result.keywordremover.AllKeywordsRemover` `method`), 380  
`end_break()` (`robot.result.keywordremover.ByTagNameKeywordRemover` `method`), 388  
`end_break()` (`robot.result.keywordremover.ByTagKeywordRemover` `method`), 392  
`end_break()` (`robot.result.keywordremover.ForLoopItemsRemover` `method`), 396  
`end_break()` (`robot.result.keywordremover.PassedKeywordRemover` `method`), 384  
`end_break()` (`robot.result.keywordremover.WaitUntilKeywordSucceedsRemover` `method`), 404  
`end_break()` (`robot.result.keywordremover.WarningAndErrorFinder` `method`), 409  
`end_break()` (`robot.result.keywordremover.WhileLoopItemsRemover` `method`), 400  
`end_break()` (`robot.result.merger.Merger` `method`), 413  
`end_break()` (`robot.result.messagefilter.MessageFilter` `method`), 417  
`end_break()` (`robot.result.resultbuilder.RemoveKeywords` `method`), 465  
`end_break()` (`robot.result.suiteteardownfailed.SuiteTeardownFailed` `method`), 473  
`end_break()` (`robot.result.suiteteardownfailed.SuiteTeardownFailureHandler` `method`), 469  
`end_break()` (`robot.result.visitor.ResultVisitor` `method`), 478  
`end_break()` (`robot.running.randomizer.Randomizer` `method`), 541



`end_break()` (`robot.running.suiterunner.SuiteRunner` method), 547  
`end_col_offset` (`robot.parsing.lexer.tokens.END` attribute), 305  
`end_col_offset` (`robot.parsing.lexer.tokens.EOS` attribute), 303  
`end_col_offset` (`robot.parsing.lexer.tokens.Token` attribute), 301  
`end_col_offset` (`robot.parsing.model.blocks.Block` attribute), 306  
`end_col_offset` (`robot.parsing.model.blocks.CommentSection` attribute), 308  
`end_col_offset` (`robot.parsing.model.blocks.File` attribute), 306  
`end_col_offset` (`robot.parsing.model.blocks.For` attribute), 309  
`end_col_offset` (`robot.parsing.model.blocks.HeaderAndBody` attribute), 306  
`end_col_offset` (`robot.parsing.model.blocks.If` attribute), 309  
`end_col_offset` (`robot.parsing.model.blocks.Keyword` attribute), 308  
`end_col_offset` (`robot.parsing.model.blocks.KeywordSection` attribute), 308  
`end_col_offset` (`robot.parsing.model.blocks.Section` attribute), 307  
`end_col_offset` (`robot.parsing.model.blocks.SettingSection` attribute), 307  
`end_col_offset` (`robot.parsing.model.blocks.TestCase` attribute), 308  
`end_col_offset` (`robot.parsing.model.blocks.TestCaseSection` attribute), 307  
`end_col_offset` (`robot.parsing.model.blocks.Try` attribute), 309  
`end_col_offset` (`robot.parsing.model.blocks.VariableSection` attribute), 307  
`end_col_offset` (`robot.parsing.model.blocks.While` attribute), 310  
`end_col_offset` (`robot.parsing.model.statements.Arguments` attribute), 334  
`end_col_offset` (`robot.parsing.model.statements.Break` attribute), 349  
`end_col_offset` (`robot.parsing.model.statements.Comment` attribute), 350  
`end_col_offset` (`robot.parsing.model.statements.Continue` attribute), 348  
`end_col_offset` (`robot.parsing.model.statements.DefaultTags` attribute), 321  
`end_col_offset` (`robot.parsing.model.statements.Documentation` attribute), 319  
`end_col_offset` (`robot.parsing.model.statements.DocumentationOrMetadata` attribute), 312  
`end_col_offset` (`robot.parsing.model.statements.ElseHeader` attribute), 341  
`end_col_offset` (`robot.parsing.model.statements.ElseIfHeader` attribute), 340  
`end_col_offset` (`robot.parsing.model.statements.EmptyLine` attribute), 352  
`end_col_offset` (`robot.parsing.model.statements.End` attribute), 345  
`end_col_offset` (`robot.parsing.model.statements.Error` attribute), 351  
`end_col_offset` (`robot.parsing.model.statements.ExceptHeader` attribute), 344  
`end_col_offset` (`robot.parsing.model.statements.FinallyHeader` attribute), 344  
`end_col_offset` (`robot.parsing.model.statements.Fixture` attribute), 314  
`end_col_offset` (`robot.parsing.model.statements.ForceTags` attribute), 320  
`end_col_offset` (`robot.parsing.model.statements.ForHeader` attribute), 337  
`end_col_offset` (`robot.parsing.model.statements.IfElseHeader` attribute), 338  
`end_col_offset` (`robot.parsing.model.statements.IfHeader` attribute), 339  
`end_col_offset` (`robot.parsing.model.statements.InlineIfHeader` attribute), 339  
`end_col_offset` (`robot.parsing.model.statements.KeywordCall` attribute), 335  
`end_col_offset` (`robot.parsing.model.statements.KeywordName` attribute), 329  
`end_col_offset` (`robot.parsing.model.statements.LibraryImport` attribute), 316  
`end_col_offset` (`robot.parsing.model.statements.LoopControl` attribute), 348  
`end_col_offset` (`robot.parsing.model.statements.Metadata` attribute), 320  
`end_col_offset` (`robot.parsing.model.statements.MultiValue` attribute), 313  
`end_col_offset` (`robot.parsing.model.statements.NoArgumentHeader` attribute), 342  
`end_col_offset` (`robot.parsing.model.statements.ResourceImport` attribute), 317  
`end_col_offset` (`robot.parsing.model.statements.Return` attribute), 334  
`end_col_offset` (`robot.parsing.model.statements.ReturnStatement` attribute), 347  
`end_col_offset` (`robot.parsing.model.statements.SectionHeader` attribute), 315  
`end_col_offset` (`robot.parsing.model.statements.Setup` attribute), 330  
`end_col_offset` (`robot.parsing.model.statements.SingleValue` attribute), 313  
`end_col_offset` (`robot.parsing.model.statements.Statement` attribute), 311  
`end_col_offset` (`robot.parsing.model.statements.SuiteSetup` attribute), 322

`end_col_offset (robot.parsing.model.statements.SuiteTeardown attribute), 323`  
`end_col_offset (robot.parsing.model.statements.Tags attribute), 331`  
`end_col_offset (robot.parsing.model.statements.Teardown attribute), 330`  
`end_col_offset (robot.parsing.model.statements.Template attribute), 332`  
`end_col_offset (robot.parsing.model.statements.TemplateArguments attribute), 336`  
`end_col_offset (robot.parsing.model.statements.TestCaseName attribute), 328`  
`end_col_offset (robot.parsing.model.statements.TestSetup attribute), 324`  
`end_col_offset (robot.parsing.model.statements.TestTeardown attribute), 325`  
`end_col_offset (robot.parsing.model.statements.TestTemplate attribute), 325`  
`end_col_offset (robot.parsing.model.statements.TestTimeout attribute), 326`  
`end_col_offset (robot.parsing.model.statements.Timeout attribute), 333`  
`end_col_offset (robot.parsing.model.statements.TryHeader attribute), 343`  
`end_col_offset (robot.parsing.model.statements.Variable attribute), 327`  
`end_col_offset (robot.parsing.model.statements.VariableImport attribute), 318`  
`end_col_offset (robot.parsing.model.statements.WhileHeader attribute), 346`  
`end_continue () (robot.conf.gatherfailed.GatherFailedSuites method), 27`  
`end_continue () (robot.conf.gatherfailed.GatherFailedTests method), 23`  
`end_continue () (robot.model.configurer.SuiteConfigurer method), 199`  
`end_continue () (robot.model.filter.EmptySuiteRemover method), 215`  
`end_continue () (robot.model.filter.Filter method), 220`  
`end_continue () (robot.model.modifier.ModelModifier method), 231`  
`end_continue () (robot.model.statistics.StatisticsBuilder method), 236`  
`end_continue () (robot.model.tagsetter.TagSetter method), 243`  
`end_continue () (robot.model.totalstatistics.TotalStatisticsBuilder method), 254`  
`end_continue () (robot.model.visitor.SuiteVisitor method), 262`  
`end_continue () (robot.output.console.dotted.StatusReporter method), 264`  
`end_continue () (robot.output.xmllogger.XmlLogger method), 280`  
`end_directory () (robot.parsing.suitestructure.SuiteStructureVisitor method), 357`  
`end_directory () (robot.running.builder.builders.SuiteStructureParser method), 504`  
`end_errors () (robot.output.xmllogger.XmlLogger method), 281`  
`end_errors () (robot.reporting.outputwriter.OutputWriter method), 360`  
`end_errors () (robot.reporting.xunitwriter.XUnitFileWriter method), 367`  
`end_errors () (robot.result.configurer.SuiteConfigurer method), 372`  
`end_errors () (robot.result.keywordremover.AllKeywordsRemover method), 380`  
`end_errors () (robot.result.keywordremover.ByNameKeywordRemover method), 388`  
`end_errors () (robot.result.keywordremover.ByTagKeywordRemover method), 392`  
`end_errors () (robot.result.keywordremover.ForLoopItemsRemover method), 396`  
`end_errors () (robot.result.keywordremover.PassedKeywordRemover method), 384`  
`end_errors () (robot.result.keywordremover.WaitUntilKeywordSucceeds method), 405`  
`end_errors () (robot.result.keywordremover.WarningAndErrorFinder method), 409`  
`end_errors () (robot.result.keywordremover.WhileLoopItemsRemover method), 400`  
`end_errors () (robot.result.merger.Merger method), 413`  
`end_errors () (robot.result.messagefilter.MessageFilter method), 418`  
`end_errors () (robot.result.resultbuilder.RemoveKeywords method), 465`  
`end_errors () (robot.result.suiteteardownfailed.SuiteTeardownFailed method), 474`  
`end_errors () (robot.result.suiteteardownfailed.SuiteTeardownFailure method), 469`  
`end_errors () (robot.result.visitor.ResultVisitor method), 478`  
`end_errors () (robot.running.randomizer.Randomizer method), 541`  
`end_errors () (robot.running.suiterunner.SuiteRunner method), 547`  
`end_for () (robot.conf.gatherfailed.GatherFailedSuites method), 27`  
`end_for () (robot.conf.gatherfailed.GatherFailedTests method), 23`

---

<code>end_for()</code> ( <i>robot.model.configurer.SuiteConfigurer</i> method), 199	<code>end_for()</code> ( <i>robot.running.suiterunner.SuiteRunner</i> method), 547
<code>end_for()</code> ( <i>robot.model.filter.EmptySuiteRemover</i> method), 216	<code>end_for_iteration()</code> ( <i>robot.conf.gatherfailed.GatherFailedSuites</i> method), 27
<code>end_for()</code> ( <i>robot.model.filter.Filter</i> method), 220	<code>end_for_iteration()</code> ( <i>robot.conf.gatherfailed.GatherFailedTests</i> method), 23
<code>end_for()</code> ( <i>robot.model.modifier.ModelModifier</i> method), 231	<code>end_for_iteration()</code> ( <i>robot.model.configurer.SuiteConfigurer</i> method), 200
<code>end_for()</code> ( <i>robot.model.statistics.StatisticsBuilder</i> method), 236	<code>end_for_iteration()</code> ( <i>robot.model.filter.EmptySuiteRemover</i> method), 216
<code>end_for()</code> ( <i>robot.model.tagsetter.TagSetter</i> method), 243	<code>end_for_iteration()</code> ( <i>robot.model.filter.Filter</i> method), 220
<code>end_for()</code> ( <i>robot.model.totalstatistics.TotalStatisticsBuilder</i> method), 254	<code>end_for_iteration()</code> ( <i>robot.model.modifier.ModelModifier</i> method), 231
<code>end_for()</code> ( <i>robot.model.visitor.SuiteVisitor</i> method), 260	<code>end_for_iteration()</code> ( <i>robot.model.statistics.StatisticsBuilder</i> method), 236
<code>end_for()</code> ( <i>robot.output.console.dotted.StatusReporter</i> method), 264	<code>end_for_iteration()</code> ( <i>robot.model.tagsetter.TagSetter</i> method), 243
<code>end_for()</code> ( <i>robot.output.xmllogger.XmlLogger</i> method), 279	<code>end_for_iteration()</code> ( <i>robot.model.totalstatistics.TotalStatisticsBuilder</i> method), 254
<code>end_for()</code> ( <i>robot.reporting.outputwriter.OutputWriter</i> method), 360	<code>end_for_iteration()</code> ( <i>robot.model.visitor.SuiteVisitor</i> method), 260
<code>end_for()</code> ( <i>robot.reporting.xunitwriter.XUnitFileWriter</i> method), 367	<code>end_for_iteration()</code> ( <i>robot.output.console.dotted.StatusReporter</i> method), 264
<code>end_for()</code> ( <i>robot.result.configurer.SuiteConfigurer</i> method), 373	<code>end_for_iteration()</code> ( <i>robot.output.xmllogger.XmlLogger</i> method), 279
<code>end_for()</code> ( <i>robot.result.keywordremover.AllKeywordsRemover</i> method), 380	<code>end_for_iteration()</code> ( <i>robot.reporting.outputwriter.OutputWriter</i> method), 360
<code>end_for()</code> ( <i>robot.result.keywordremover.ByNameKeywordRemover</i> method), 388	<code>end_for_iteration()</code> ( <i>robot.reporting.xunitwriter.XUnitFileWriter</i> method), 367
<code>end_for()</code> ( <i>robot.result.keywordremover.ByTagKeywordRemover</i> method), 392	<code>end_for_iteration()</code> ( <i>robot.result.configurer.SuiteConfigurer</i> method), 373
<code>end_for()</code> ( <i>robot.result.keywordremover.ForLoopItemsRemover</i> method), 396	<code>end_for_iteration()</code> ( <i>robot.result.keywordremover.AllKeywordsRemover</i> method), 380
<code>end_for()</code> ( <i>robot.result.keywordremover.PassedKeywordRemover</i> method), 384	<code>end_for_iteration()</code> ( <i>robot.result.keywordremover.ByNameKeywordRemover</i> method), 388
<code>end_for()</code> ( <i>robot.result.keywordremover.WaitUntilKeywordSucceededKeywordRemover</i> method), 405	<code>end_for_iteration()</code> ( <i>robot.result.keywordremover.ByTagKeywordRemover</i> method), 392
<code>end_for()</code> ( <i>robot.result.keywordremover.WarningAndErrorFinder</i> method), 409	<code>end_for_iteration()</code> ( <i>robot.result.keywordremover.ForLoopItemsRemover</i> method), 396
<code>end_for()</code> ( <i>robot.result.keywordremover.WhileLoopItemsRemover</i> method), 401	<code>end_for_iteration()</code> ( <i>robot.result.keywordremover.PassedKeywordRemover</i> method), 384
<code>end_for()</code> ( <i>robot.result.merger.Merger</i> method), 413	<code>end_for_iteration()</code> ( <i>robot.result.keywordremover.WaitUntilKeywordSucceededKeywordRemover</i> method), 405
<code>end_for()</code> ( <i>robot.result.messagefilter.MessageFilter</i> method), 418	<code>end_for_iteration()</code> ( <i>robot.result.keywordremover.WarningAndErrorFinder</i> method), 409
<code>end_for()</code> ( <i>robot.result.resultbuilder.RemoveKeywords</i> method), 465	<code>end_for_iteration()</code> ( <i>robot.result.keywordremover.WhileLoopItemsRemover</i> method), 401
<code>end_for()</code> ( <i>robot.result.suiteteardownfailed.SuiteTeardownFailed</i> method), 474	<code>end_for_iteration()</code> ( <i>robot.result.merger.Merger</i> method), 413
<code>end_for()</code> ( <i>robot.result.suiteteardownfailed.SuiteTeardownFailureHandler</i> method), 469	<code>end_for_iteration()</code> ( <i>robot.result.messagefilter.MessageFilter</i> method), 418
<code>end_for()</code> ( <i>robot.result.visitor.ResultVisitor</i> method), 479	<code>end_for_iteration()</code> ( <i>robot.result.resultbuilder.RemoveKeywords</i> method), 465
<code>end_for()</code> ( <i>robot.running.randomizer.Randomizer</i> method), 541	<code>end_for_iteration()</code> ( <i>robot.result.suiteteardownfailed.SuiteTeardownFailed</i> method), 474
	<code>end_for_iteration()</code> ( <i>robot.result.suiteteardownfailed.SuiteTeardownFailureHandler</i> method), 469
	<code>end_for_iteration()</code> ( <i>robot.result.visitor.ResultVisitor</i> method), 479
	<code>end_for_iteration()</code> ( <i>robot.running.randomizer.Randomizer</i> method), 541

*method*), 392  
end\_for\_iteration() (*robot.result.keywordremover.ForLoopItemsRemover* *method*), 396  
end\_for\_iteration() (*robot.result.keywordremover.PassedKeywordRemover* *method*), 384  
end\_for\_iteration() (*robot.result.keywordremover.WaitUntilKeywordSucceedsRemover* *method*), 405  
end\_for\_iteration() (*robot.result.keywordremover.WarningAndErrorFinder* *method*), 409  
end\_for\_iteration() (*robot.result.keywordremover.WhileLoopItemsRemover* *method*), 401  
end\_for\_iteration() (*robot.result.merger.Merger* *method*), 414  
end\_for\_iteration() (*robot.result.messagefilter.MessageFilter* *method*), 418  
end\_for\_iteration() (*robot.result.resultbuilder.RemoveKeywords* *method*), 465  
end\_for\_iteration() (*robot.result.suiteteardownfailed.SuiteTeardownFailed* *method*), 474  
end\_for\_iteration() (*robot.result.suiteteardownfailed.SuiteTeardownFailureHandler* *method*), 470  
end\_for\_iteration() (*robot.result.visitor.ResultVisitor* *method*), 479  
end\_for\_iteration() (*robot.running.randomizer.Randomizer* *method*), 541  
end\_for\_iteration() (*robot.running.suiterunner.SuiteRunner* *method*), 547  
end\_if() (*robot.conf.gatherfailed.GatherFailedSuites* *method*), 28  
end\_if() (*robot.conf.gatherfailed.GatherFailedTests* *method*), 23  
end\_if() (*robot.model.configurer.SuiteConfigurer* *method*), 200  
end\_if() (*robot.model.filter.EmptySuiteRemover* *method*), 216  
end\_if() (*robot.model.filter.Filter* *method*), 220  
end\_if() (*robot.model.modifier.ModelModifier* *method*), 231  
end\_if() (*robot.model.statistics.StatisticsBuilder* *method*), 236  
end\_if() (*robot.model.tagsetter.TagSetter* *method*), 243  
end\_if() (*robot.model.totalstatistics.TotalStatisticsBuilder* *method*), 254  
end\_if() (*robot.model.visitor.SuiteVisitor* *method*), 260  
end\_if() (*robot.output.console.dotted.StatusReporter* *method*), 264  
end\_if() (*robot.output.xmllogger.XmlLogger* *method*), 278  
end\_if() (*robot.reporting.outputwriter.OutputWriter* *method*), 360  
end\_if() (*robot.reporting.xunitwriter.XUnitFileWriter* *method*), 367  
end\_if() (*robot.result.configurer.SuiteConfigurer* *method*), 373  
end\_if() (*robot.result.keywordremover.AllKeywordsRemover* *method*), 380  
end\_if() (*robot.result.keywordremover.ByNameKeywordRemover* *method*), 388  
end\_if() (*robot.result.keywordremover.ByTagKeywordRemover* *method*), 392  
end\_if() (*robot.result.keywordremover.ForLoopItemsRemover* *method*), 396  
end\_if() (*robot.result.keywordremover.PassedKeywordRemover* *method*), 384  
end\_if() (*robot.result.keywordremover.WaitUntilKeywordSucceedsRemover* *method*), 405  
end\_if() (*robot.result.keywordremover.WarningAndErrorFinder* *method*), 409  
end\_if() (*robot.result.keywordremover.WhileLoopItemsRemover* *method*), 401  
end\_if() (*robot.result.merger.Merger* *method*), 414  
end\_if() (*robot.result.messagefilter.MessageFilter* *method*), 418  
end\_if() (*robot.result.resultbuilder.RemoveKeywords* *method*), 465  
end\_if() (*robot.result.suiteteardownfailed.SuiteTeardownFailed* *method*), 474  
end\_if() (*robot.result.suiteteardownfailed.SuiteTeardownFailureHandler* *method*), 470  
end\_if() (*robot.result.visitor.ResultVisitor* *method*), 479  
end\_if() (*robot.running.randomizer.Randomizer* *method*), 541  
end\_if() (*robot.running.suiterunner.SuiteRunner* *method*), 547  
end\_if\_branch() (*robot.conf.gatherfailed.GatherFailedSuites* *method*), 28  
end\_if\_branch() (*robot.conf.gatherfailed.GatherFailedTests* *method*), 23  
end\_if\_branch() (*robot.model.configurer.SuiteConfigurer* *method*), 200  
end\_if\_branch() (*robot.model.filter.EmptySuiteRemover* *method*), 216  
end\_if\_branch() (*robot.model.filter.Filter* *method*),



220  
 end\_if\_branch() (*robot.model.modifier.ModelModifier* end\_keyword() (*robot.conf.gatherfailed.GatherFailedTests*  
   *method*), 231 *method*), 23  
 end\_if\_branch() (*robot.model.statistics.StatisticsBuilder* end\_keyword() (*robot.model.configurer.SuiteConfigurer*  
   *method*), 236 *method*), 200  
 end\_if\_branch() (*robot.model.tagsetter.TagSetter* end\_keyword() (*robot.model.filter.EmptySuiteRemover*  
   *method*), 243 *method*), 216  
 end\_if\_branch() (*robot.model.totalstatistics.TotalStatisticsBuilder* end\_keyword() (*robot.model.filter.Filter* *method*),  
   *method*), 254 220  
 end\_if\_branch() (*robot.model.visitor.SuiteVisitor* end\_keyword() (*robot.model.modifier.ModelModifier*  
   *method*), 261 *method*), 231  
 end\_if\_branch() (*robot.output.console.dotted.StatusReporter* end\_keyword() (*robot.model.statistics.StatisticsBuilder*  
   *method*), 264 *method*), 236  
 end\_if\_branch() (*robot.output.xmllogger.XmlLogger* end\_keyword() (*robot.model.tagsetter.TagSetter*  
   *method*), 279 *method*), 243  
 end\_if\_branch() (*robot.reporting.outputwriter.OutputWriter* end\_keyword() (*robot.model.totalstatistics.TotalStatisticsBuilder*  
   *method*), 360 *method*), 254  
 end\_if\_branch() (*robot.reporting.xunitwriter.XUnitFileWriter* end\_keyword() (*robot.model.visitor.SuiteVisitor*  
   *method*), 367 *method*), 260  
 end\_if\_branch() (*robot.result.configurer.SuiteConfigurer* end\_keyword() (*robot.output.console.dotted.StatusReporter*  
   *method*), 373 *method*), 264  
 end\_if\_branch() (*robot.result.keywordremover.AllKeywordsRemover* end\_keyword() (*robot.output.console.verbose.VerboseOutput*  
   *method*), 380 *method*), 269  
 end\_if\_branch() (*robot.result.keywordremover.ByNameKeywordRemover* end\_keyword() (*robot.output.filelogger.FileLogger*  
   *method*), 388 *method*), 270  
 end\_if\_branch() (*robot.result.keywordremover.ByTagKeywordRemover* end\_keyword() (*robot.output.listeners.Listeners*  
   *method*), 392 *method*), 272  
 end\_if\_branch() (*robot.result.keywordremover.ForLoopItemsRemover* end\_keyword() (*robot.output.logger.Logger* *method*),  
   *method*), 397 273  
 end\_if\_branch() (*robot.result.keywordremover.PassedKeywordRemover* end\_keyword() (*robot.output.logger.LoggerProxy*  
   *method*), 384 *method*), 274  
 end\_if\_branch() (*robot.result.keywordremover.WaitUntilKeywordSucceedsRemover* end\_keyword() (*robot.output.output.Output* *method*),  
   *method*), 405 276  
 end\_if\_branch() (*robot.result.keywordremover.WarningAndErrorFinder* end\_keyword() (*robot.output.xmllogger.XmlLogger*  
   *method*), 409 *method*), 278  
 end\_if\_branch() (*robot.result.keywordremover.WhileLoopItemsRemover* end\_keyword() (*robot.reporting.outputwriter.OutputWriter*  
   *method*), 401 *method*), 360  
 end\_if\_branch() (*robot.result.merger.Merger* end\_keyword() (*robot.reporting.xunitwriter.XUnitFileWriter*  
   *method*), 414 *method*), 367  
 end\_if\_branch() (*robot.result.messagefilter.MessageFilter* end\_keyword() (*robot.result.configurer.SuiteConfigurer*  
   *method*), 418 *method*), 373  
 end\_if\_branch() (*robot.result.resultbuilder.RemoveKeywords* end\_keyword() (*robot.result.keywordremover.AllKeywordsRemover*  
   *method*), 465 *method*), 380  
 end\_if\_branch() (*robot.result.suitetardownfailed.SuiteTeardownFailed* end\_keyword() (*robot.result.keywordremover.ByNameKeywordRemover*  
   *method*), 474 *method*), 388  
 end\_if\_branch() (*robot.result.suitetardownfailed.SuiteTeardownFailedHandler* end\_keyword() (*robot.result.keywordremover.ByTagKeywordRemover*  
   *method*), 470 *method*), 392  
 end\_if\_branch() (*robot.result.visitor.ResultVisitor* end\_keyword() (*robot.result.keywordremover.ForLoopItemsRemover*  
   *method*), 479 *method*), 397  
 end\_if\_branch() (*robot.running.randomizer.Randomizer* end\_keyword() (*robot.result.keywordremover.PassedKeywordRemover*  
   *method*), 541 *method*), 384  
 end\_if\_branch() (*robot.running.suiterunner.SuiteRunner* end\_keyword() (*robot.result.keywordremover.WaitUntilKeywordSucceedsRemover*  
   *method*), 547 *method*), 405  
 end\_keyword() (*robot.conf.gatherfailed.GatherFailedSuite* end\_keyword() (*robot.result.keywordremover.WarningAndErrorFinder*

`method`), 410  
`end_keyword()` (`robot.result.keywordremover.WhileLoopItemsRemover` `method`), 401  
`end_keyword()` (`robot.result.merger.Merger` `method`), 414  
`end_keyword()` (`robot.result.messagefilter.MessageFilter` `method`), 418  
`end_keyword()` (`robot.result.resultbuilder.RemoveKeywords` `method`), 465  
`end_keyword()` (`robot.result.suiteteardownfailed.SuiteTeardownFailed` `method`), 474  
`end_keyword()` (`robot.result.suiteteardownfailed.SuiteTeardownFailureHandler` `method`), 470  
`end_keyword()` (`robot.result.visitor.ResultVisitor` `method`), 479  
`end_keyword()` (`robot.running.randomizer.Randomizer` `method`), 541  
`end_keyword()` (`robot.running.suiterunner.SuiteRunner` `method`), 548  
`end_keyword()` (`robot.variables.scopes.SetVariables` `method`), 572  
`end_keyword()` (`robot.variables.scopes.VariableScopes` `method`), 571  
`end_lineno` (`robot.parsing.model.blocks.Block` `attribute`), 306  
`end_lineno` (`robot.parsing.model.blocks.CommentSection` `attribute`), 308  
`end_lineno` (`robot.parsing.model.blocks.File` `attribute`), 306  
`end_lineno` (`robot.parsing.model.blocks.For` `attribute`), 309  
`end_lineno` (`robot.parsing.model.blocks.HeaderAndBody` `attribute`), 306  
`end_lineno` (`robot.parsing.model.blocks.If` `attribute`), 309  
`end_lineno` (`robot.parsing.model.blocks.Keyword` `attribute`), 308  
`end_lineno` (`robot.parsing.model.blocks.KeywordSection` `attribute`), 308  
`end_lineno` (`robot.parsing.model.blocks.Section` `attribute`), 307  
`end_lineno` (`robot.parsing.model.blocks.SettingSection` `attribute`), 307  
`end_lineno` (`robot.parsing.model.blocks.TestCase` `attribute`), 308  
`end_lineno` (`robot.parsing.model.blocks.TestCaseSection` `attribute`), 307  
`end_lineno` (`robot.parsing.model.blocks.Try` `attribute`), 309  
`end_lineno` (`robot.parsing.model.blocks.VariableSection` `attribute`), 307  
`end_lineno` (`robot.parsing.model.blocks.While` `attribute`), 310  
`end_lineno` (`robot.parsing.model.statements.Arguments` `attribute`), 334  
`end_lineno` (`robot.parsing.model.statements.Break` `attribute`), 349  
`end_lineno` (`robot.parsing.model.statements.Comment` `attribute`), 350  
`end_lineno` (`robot.parsing.model.statements.Continue` `attribute`), 348  
`end_lineno` (`robot.parsing.model.statements.DefaultTags` `attribute`), 321  
`end_lineno` (`robot.parsing.model.statements.Documentation` `attribute`), 319  
`end_lineno` (`robot.parsing.model.statements.DocumentationOrMetadata` `attribute`), 312  
`end_lineno` (`robot.parsing.model.statements.ElseHeader` `attribute`), 341  
`end_lineno` (`robot.parsing.model.statements.ElseIfHeader` `attribute`), 340  
`end_lineno` (`robot.parsing.model.statements.EmptyLine` `attribute`), 352  
`end_lineno` (`robot.parsing.model.statements.End` `attribute`), 345  
`end_lineno` (`robot.parsing.model.statements.Error` `attribute`), 351  
`end_lineno` (`robot.parsing.model.statements.ExceptHeader` `attribute`), 344  
`end_lineno` (`robot.parsing.model.statements.FinallyHeader` `attribute`), 344  
`end_lineno` (`robot.parsing.model.statements.Fixture` `attribute`), 314  
`end_lineno` (`robot.parsing.model.statements.ForceTags` `attribute`), 320  
`end_lineno` (`robot.parsing.model.statements.ForHeader` `attribute`), 337  
`end_lineno` (`robot.parsing.model.statements.IfElseHeader` `attribute`), 338  
`end_lineno` (`robot.parsing.model.statements.IfHeader` `attribute`), 339  
`end_lineno` (`robot.parsing.model.statements.InlineIfHeader` `attribute`), 339  
`end_lineno` (`robot.parsing.model.statements.KeywordCall` `attribute`), 335  
`end_lineno` (`robot.parsing.model.statements.KeywordName` `attribute`), 329  
`end_lineno` (`robot.parsing.model.statements.LibraryImport` `attribute`), 316  
`end_lineno` (`robot.parsing.model.statements.LoopControl` `attribute`), 348  
`end_lineno` (`robot.parsing.model.statements.Metadata` `attribute`), 320  
`end_lineno` (`robot.parsing.model.statements.MultiValue` `attribute`), 313  
`end_lineno` (`robot.parsing.model.statements.NoArgumentHeader` `attribute`), 342  
`end_lineno` (`robot.parsing.model.statements.ResourceImport`

`attribute`), 317  
`end_lineno (robot.parsing.model.statements.ReturnState`  
`attribute`), 334  
`end_lineno (robot.parsing.model.statements.ReturnState`  
`attribute`), 347  
`end_lineno (robot.parsing.model.statements.SectionHeader`  
`attribute`), 315  
`end_lineno (robot.parsing.model.statements.Setup at-`  
`tribute`), 330  
`end_lineno (robot.parsing.model.statements.SingleValue`  
`attribute`), 313  
`end_lineno (robot.parsing.model.statements.Statement`  
`attribute`), 311  
`end_lineno (robot.parsing.model.statements.SuiteSetup`  
`attribute`), 322  
`end_lineno (robot.parsing.model.statements.SuiteTeardown`  
`attribute`), 323  
`end_lineno (robot.parsing.model.statements.Tags at-`  
`tribute`), 331  
`end_lineno (robot.parsing.model.statements.Teardown`  
`attribute`), 330  
`end_lineno (robot.parsing.model.statements.Template`  
`attribute`), 332  
`end_lineno (robot.parsing.model.statements.TemplateArguments`  
`attribute`), 336  
`end_lineno (robot.parsing.model.statements.TestCaseName`  
`attribute`), 328  
`end_lineno (robot.parsing.model.statements.TestSetup`  
`attribute`), 324  
`end_lineno (robot.parsing.model.statements.TestTeardown`  
`attribute`), 325  
`end_lineno (robot.parsing.model.statements.TestTemplate`  
`attribute`), 325  
`end_lineno (robot.parsing.model.statements.TestTimeout`  
`attribute`), 326  
`end_lineno (robot.parsing.model.statements.Timeout`  
`attribute`), 333  
`end_lineno (robot.parsing.model.statements.TryHeader`  
`attribute`), 343  
`end_lineno (robot.parsing.model.statements.Variable`  
`attribute`), 327  
`end_lineno (robot.parsing.model.statements.VariablesImport`  
`attribute`), 318  
`end_lineno (robot.parsing.model.statements.WhileHeader`  
`attribute`), 346  
`end_loggers (robot.output.logger.Logger attribute),`  
273  
`end_message () (robot.conf.gatherfailed.GatherFailedSuite`  
`method`), 28  
`end_message () (robot.conf.gatherfailed.GatherFailedTest`  
`method`), 24  
`end_message () (robot.model.configurer.SuiteConfigurer`  
`method`), 200  
`end_message () (robot.model.filter.EmptySuiteRemover`  
`method`), 216  
`end_message () (robot.model.filter.Filter method),`  
220  
`end_message () (robot.model.modifier.ModelModifier`  
`method`), 231  
`end_message () (robot.model.statistics.StatisticsBuilder`  
`method`), 236  
`end_message () (robot.model.tagsetter.TagSetter`  
`method`), 243  
`end_message () (robot.model.totalstatistics.TotalStatisticsBuilder`  
`method`), 254  
`end_message () (robot.model.visitor.SuiteVisitor`  
`method`), 263  
`end_message () (robot.output.console.dotted.StatusReporter`  
`method`), 264  
`end_message () (robot.output.xmllogger.XmlLogger`  
`method`), 281  
`end_message () (robot.reporting.outputwriter.OutputWriter`  
`method`), 360  
`end_message () (robot.reporting.xunitwriter.XUnitFileWriter`  
`method`), 367  
`end_message () (robot.result.configurer.SuiteConfigurer`  
`method`), 373  
`end_message () (robot.result.keywordremover.AllKeywordsRemover`  
`method`), 380  
`end_message () (robot.result.keywordremover.ByNameKeywordRemover`  
`method`), 388  
`end_message () (robot.result.keywordremover.ByTagKeywordRemover`  
`method`), 393  
`end_message () (robot.result.keywordremover.ForLoopItemsRemover`  
`method`), 397  
`end_message () (robot.result.keywordremover.PassedKeywordRemover`  
`method`), 384  
`end_message () (robot.result.keywordremover.WaitUntilKeywordSucceed`  
`method`), 405  
`end_message () (robot.result.keywordremover.WarningAndErrorFinder`  
`method`), 410  
`end_message () (robot.result.keywordremover.WhileLoopItemsRemover`  
`method`), 401  
`end_message () (robot.result.merger.Merger method),`  
414  
`end_message () (robot.result.messagefilter.MessageFilter`  
`method`), 418  
`end_message () (robot.result.resultbuilder.RemoveKeywords`  
`method`), 465  
`end_message () (robot.result.suite teardownfailed.SuiteTeardownFailed`  
`method`), 474  
`end_message () (robot.result.suite teardownfailed.SuiteTeardownFailure`  
`method`), 470  
`end_message () (robot.result.visitor.ResultVisitor`  
`method`), 479  
`end_message () (robot.running.randomizer.Randomizer`  
`method`), 541  
`end_message () (robot.running.suiterunner.SuiteRunner`

method), 548  
 end\_result() (robot.output.xmllogger.XmlLogger method), 281  
 end\_result() (robot.reporting.outputwriter.OutputWriter method), 360  
 end\_result() (robot.reporting.xunitwriter.XUnitFileWriter method), 366  
 end\_result() (robot.result.visitor.ResultVisitor method), 478  
 end\_return() (robot.conf.gatherfailed.GatherFailedSuites method), 28  
 end\_return() (robot.conf.gatherfailed.GatherFailedTests method), 24  
 end\_return() (robot.model.configurer.SuiteConfigurer method), 200  
 end\_return() (robot.model.filter.EmptySuiteRemover method), 216  
 end\_return() (robot.model.filter.Filter method), 220  
 end\_return() (robot.model.modifier.ModelModifier method), 231  
 end\_return() (robot.model.statistics.StatisticsBuilder method), 236  
 end\_return() (robot.model.tagsetter.TagSetter method), 243  
 end\_return() (robot.model.totalstatistics.TotalStatisticsBuilder method), 254  
 end\_return() (robot.model.visitor.SuiteVisitor method), 262  
 end\_return() (robot.output.console.dotted.StatusReporter method), 265  
 end\_return() (robot.output.xmllogger.XmlLogger method), 280  
 end\_return() (robot.reporting.outputwriter.OutputWriter method), 360  
 end\_return() (robot.reporting.xunitwriter.XUnitFileWriter method), 367  
 end\_return() (robot.result.configurer.SuiteConfigurer method), 373  
 end\_return() (robot.result.keywordremover.AllKeywordsRemover method), 380  
 end\_return() (robot.result.keywordremover.ByNameKeywordsRemover method), 388  
 end\_return() (robot.result.keywordremover.ByTagKeywordsRemover method), 393  
 end\_return() (robot.result.keywordremover.ForLoopItemsRemover method), 397  
 end\_return() (robot.result.keywordremover.PassedKeywordRemover method), 384  
 end\_return() (robot.result.keywordremover.WaitUntilKeywordSucceeds method), 405  
 end\_return() (robot.result.keywordremover.WarningAndErrorFinder method), 410  
 end\_return() (robot.result.keywordremover.WhileLoopItemsRemover method), 401  
 end\_return() (robot.result.merger.Merger method), 414  
 end\_return() (robot.result.messagefilter.MessageFilter method), 418  
 end\_return() (robot.result.resultbuilder.RemoveKeywords method), 466  
 end\_return() (robot.result.suitetardownfailed.SuiteTeardownFailed method), 474  
 end\_return() (robot.result.suitetardownfailed.SuiteTeardownFailureHandler method), 470  
 end\_return() (robot.result.visitor.ResultVisitor method), 479  
 end\_return() (robot.running.randomizer.Randomizer method), 542  
 end\_return() (robot.running.suiterunner.SuiteRunner method), 548  
 end\_splitting() (robot.reporting.jsbuildingcontext.JsBuildingContext method), 358  
 end\_stat() (robot.output.xmllogger.XmlLogger method), 281  
 end\_stat() (robot.reporting.outputwriter.OutputWriter method), 360  
 end\_stat() (robot.reporting.xunitwriter.XUnitFileWriter method), 367  
 end\_stat() (robot.result.visitor.ResultVisitor method), 478  
 end\_statistics() (robot.output.xmllogger.XmlLogger method), 281  
 end\_statistics() (robot.reporting.outputwriter.OutputWriter method), 360  
 end\_statistics() (robot.reporting.xunitwriter.XUnitFileWriter method), 367  
 end\_statistics() (robot.result.visitor.ResultVisitor method), 478  
 end\_suite() (robot.conf.gatherfailed.GatherFailedSuites method), 28  
 end\_suite() (robot.conf.gatherfailed.GatherFailedTests method), 24  
 end\_suite() (robot.model.configurer.SuiteConfigurer method), 200  
 end\_suite() (robot.model.filter.EmptySuiteRemover method), 215  
 end\_suite() (robot.model.filter.Filter method), 220  
 end\_suite() (robot.model.modifier.ModelModifier method), 231  
 end\_suite() (robot.model.statistics.StatisticsBuilder method), 235  
 end\_suite() (robot.model.sitestatistics.SuiteStatisticsBuilder method), 242  
 end\_suite() (robot.model.tagsetter.TagSetter method), 243  
 end\_suite() (robot.model.totalstatistics.TotalStatisticsBuilder method), 254  
 end\_suite() (robot.model.visitor.SuiteVisitor



`method`), 259  
`end_suite()` (`robot.output.console.dotted.DottedOutput` `method`), 264  
`end_suite()` (`robot.output.console.dotted.StatusReporter` `method`), 265  
`end_suite()` (`robot.output.console.verbose.VerboseOutput` `method`), 269  
`end_suite()` (`robot.output.filelogger.FileLogger` `method`), 270  
`end_suite()` (`robot.output.logger.Logger` `method`), 273  
`end_suite()` (`robot.output.output.Output` `method`), 276  
`end_suite()` (`robot.output.xmllogger.XmlLogger` `method`), 281  
`end_suite()` (`robot.reporting.outputwriter.OutputWriter` `method`), 360  
`end_suite()` (`robot.reporting.xunitwriter.XUnitFileWriter` `method`), 366  
`end_suite()` (`robot.result.configurer.SuiteConfigurer` `method`), 373  
`end_suite()` (`robot.result.keywordremover.AllKeywordsRemover` `method`), 380  
`end_suite()` (`robot.result.keywordremover.ByNameKeywordRemover` `method`), 389  
`end_suite()` (`robot.result.keywordremover.ByTagKeywordRemover` `method`), 393  
`end_suite()` (`robot.result.keywordremover.ForLoopItemsRemover` `method`), 397  
`end_suite()` (`robot.result.keywordremover.PassedKeywordRemover` `method`), 385  
`end_suite()` (`robot.result.keywordremover.WaitUntilKeywordSucceedsRemover` `method`), 405  
`end_suite()` (`robot.result.keywordremover.WarningAndErrorFinder` `method`), 410  
`end_suite()` (`robot.result.keywordremover.WhileLoopItemsRemover` `method`), 401  
`end_suite()` (`robot.result.merger.Merger` `method`), 413  
`end_suite()` (`robot.result.messagefilter.MessageFilter` `method`), 418  
`end_suite()` (`robot.result.resultbuilder.RemoveKeywords` `method`), 466  
`end_suite()` (`robot.result.suiteteardownfailed.SuiteTeardownFailed` `method`), 474  
`end_suite()` (`robot.result.suiteteardownfailed.SuiteTeardownFailedHard` `method`), 469  
`end_suite()` (`robot.result.visitor.ResultVisitor` `method`), 479  
`end_suite()` (`robot.running.context.ExecutionContexts` `method`), 513  
`end_suite()` (`robot.running.libraryscopes.GlobalScope` `method`), 515  
`end_suite()` (`robot.running.libraryscopes.TestCaseScope` `method`), 516  
`end_suite()` (`robot.running.libraryscopes.TestSuiteScope` `method`), 515  
`end_suite()` (`robot.running.namespace.Namespace` `method`), 540  
`end_suite()` (`robot.running.randomizer.Randomizer` `method`), 542  
`end_suite()` (`robot.running.suiterunner.SuiteRunner` `method`), 547  
`end_suite()` (`robot.variables.scopes.SetVariables` `method`), 571  
`end_suite()` (`robot.variables.scopes.VariableScopes` `method`), 571  
`end_suite_statistics()` (`robot.output.xmllogger.XmlLogger` `method`), 281  
`end_suite_statistics()` (`robot.reporting.outputwriter.OutputWriter` `method`), 360  
`end_suite_statistics()` (`robot.reporting.xunitwriter.XUnitFileWriter` `method`), 367  
`end_suite_statistics()` (`robot.result.visitor.ResultVisitor` `method`), 478  
`end_suite_statistics()` (`robot.output.xmllogger.XmlLogger` `method`), 281  
`end_tag_statistics()` (`robot.reporting.outputwriter.OutputWriter` `method`), 360  
`end_tag_statistics()` (`robot.reporting.xunitwriter.XUnitFileWriter` `method`), 367  
`end_tag_statistics()` (`robot.result.visitor.ResultVisitor` `method`), 478  
`end_test()` (`robot.conf.gatherfailed.GatherFailedSuites` `method`), 28  
`end_test()` (`robot.conf.gatherfailed.GatherFailedTests` `method`), 24  
`end_test()` (`robot.model.configurer.SuiteConfigurer` `method`), 200  
`end_test()` (`robot.model.filter.EmptySuiteRemover` `method`), 216  
`end_test()` (`robot.model.filter.Filter` `method`), 220  
`end_test()` (`robot.model.modifier.ModelModifier` `method`), 231  
`end_test()` (`robot.model.statistics.StatisticsBuilder` `method`), 236  
`end_test()` (`robot.model.tagsetter.TagSetter` `method`), 243  
`end_test()` (`robot.model.totalstatistics.TotalStatisticsBuilder` `method`), 254

`end_test()` (*robot.model.visitor.SuiteVisitor* method), 259  
`end_test()` (*robot.output.console.dotted.DottedOutput* method), 264  
`end_test()` (*robot.output.console.dotted.StatusReporter* method), 265  
`end_test()` (*robot.output.console.verbose.VerboseOutput* method), 269  
`end_test()` (*robot.output.filelogger.FileLogger* method), 270  
`end_test()` (*robot.output.logger.Logger* method), 273  
`end_test()` (*robot.output.output.Output* method), 276  
`end_test()` (*robot.output.xmllogger.XmlLogger* method), 280  
`end_test()` (*robot.reporting.outputwriter.OutputWriter* method), 361  
`end_test()` (*robot.reporting.xunitwriter.XUnitFileWriter* method), 367  
`end_test()` (*robot.result.configurer.SuiteConfigurer* method), 373  
`end_test()` (*robot.result.keywordremover.AllKeywordsRemover* method), 380  
`end_test()` (*robot.result.keywordremover.ByNameKeywordRemover* method), 389  
`end_test()` (*robot.result.keywordremover.ByTagKeywordRemover* method), 393  
`end_test()` (*robot.result.keywordremover.ForLoopItemsRemover* method), 397  
`end_test()` (*robot.result.keywordremover.PassedKeywordRemover* method), 385  
`end_test()` (*robot.result.keywordremover.WaitUntilKeywordsSucceedsRemover* method), 405  
`end_test()` (*robot.result.keywordremover.WarningAndErrorFinder* method), 410  
`end_test()` (*robot.result.keywordremover.WhileLoopItemsRemover* method), 401  
`end_test()` (*robot.result.merger.Merger* method), 414  
`end_test()` (*robot.result.messagefilter.MessageFilter* method), 418  
`end_test()` (*robot.result.resultbuilder.RemoveKeywords* method), 466  
`end_test()` (*robot.result.suiteteardownfailed.SuiteTeardownFailed* method), 474  
`end_test()` (*robot.result.suiteteardownfailed.SuiteTeardownFailureHandler* method), 470  
`end_test()` (*robot.result.visitor.ResultVisitor* method), 479  
`end_test()` (*robot.running.libraryscopes.GlobalScope* method), 515  
`end_test()` (*robot.running.libraryscopes.TestCaseScope* method), 515  
`end_test()` (*robot.running.libraryscopes.TestSuiteScope* method), 515  
`end_test()` (*robot.running.namespace.Namespace* method), 540  
`end_test()` (*robot.running.randomizer.Randomizer* method), 542  
`end_test()` (*robot.running.suiterunner.SuiteRunner* method), 548  
`end_test()` (*robot.variables.scopes.SetVariables* method), 571  
`end_test()` (*robot.variables.scopes.VariableScopes* method), 571  
`end_total_statistics()` (*robot.output.xmllogger.XmlLogger* method), 281  
`end_total_statistics()` (*robot.reporting.outputwriter.OutputWriter* method), 361  
`end_total_statistics()` (*robot.reporting.xunitwriter.XUnitFileWriter* method), 367  
`end_total_statistics()` (*robot.result.visitor.ResultVisitor* method), 478  
`end_try()` (*robot.conf.gatherfailed.GatherFailedSuites* method), 28  
`end_try()` (*robot.conf.gatherfailed.GatherFailedTests* method), 24  
`end_try()` (*robot.model.configurer.SuiteConfigurer* method), 200  
`end_try()` (*robot.model.filter.EmptySuiteRemover* method), 216  
`end_try()` (*robot.model.filter.Filter* method), 220  
`end_try()` (*robot.model.modifier.ModelModifier* method), 231  
`end_try()` (*robot.model.statistics.StatisticsBuilder* method), 236  
`end_try()` (*robot.model.tagsetter.TagSetter* method), 243  
`end_try()` (*robot.model.totalstatistics.TotalStatisticsBuilder* method), 254  
`end_try()` (*robot.model.visitor.SuiteVisitor* method), 261  
`end_try()` (*robot.output.console.dotted.StatusReporter* method), 265  
`end_try()` (*robot.output.xmllogger.XmlLogger* method), 279  
`end_try()` (*robot.reporting.outputwriter.OutputWriter* method), 361  
`end_try()` (*robot.reporting.xunitwriter.XUnitFileWriter* method), 367  
`end_try()` (*robot.result.configurer.SuiteConfigurer* method), 373  
`end_try()` (*robot.result.keywordremover.AllKeywordsRemover* method), 380  
`end_try()` (*robot.result.keywordremover.ByNameKeywordRemover* method), 389

[end\\_try\(\) \(robot.result.keywordremover.ByTagKeywordRemover method\), 368](#)  
[end\\_try\(\) \(robot.result.keywordremover.ForLoopItemsRemover method\), 373](#)  
[end\\_try\(\) \(robot.result.keywordremover.PassedKeywordRemover method\), 381](#)  
[end\\_try\(\) \(robot.result.keywordremover.WaitUntilKeywordSucceeded method\), 389](#)  
[end\\_try\(\) \(robot.result.keywordremover.WarningAndErrorFinder method\), 393](#)  
[end\\_try\(\) \(robot.result.keywordremover.WhileLoopItemsRemover method\), 397](#)  
[end\\_try\(\) \(robot.result.merger.Merger method\), 414](#)  
[end\\_try\(\) \(robot.result.messagefilter.MessageFilter method\), 418](#)  
[end\\_try\(\) \(robot.result.resultbuilder.RemoveKeywords method\), 466](#)  
[end\\_try\(\) \(robot.result.suiteardownfailed.SuiteTeardownFailed method\), 474](#)  
[end\\_try\(\) \(robot.result.suiteardownfailed.SuiteTeardownFailureHandler method\), 470](#)  
[end\\_try\(\) \(robot.result.visitor.ResultVisitor method\), 479](#)  
[end\\_try\(\) \(robot.running.randomizer.Randomizer method\), 542](#)  
[end\\_try\(\) \(robot.running.suiterunner.SuiteRunner method\), 548](#)  
[end\\_try\\_branch\(\) \(robot.conf.gatherfailed.GatherFailedSuites method\), 28](#)  
[end\\_try\\_branch\(\) \(robot.conf.gatherfailed.GatherFailedTests method\), 24](#)  
[end\\_try\\_branch\(\) \(robot.model.configurer.SuiteConfigurer method\), 200](#)  
[end\\_try\\_branch\(\) \(robot.model.filter.EmptySuiteRemover method\), 216](#)  
[end\\_try\\_branch\(\) \(robot.model.filter.Filter method\), 220](#)  
[end\\_try\\_branch\(\) \(robot.model.modifier.ModelModifier method\), 231](#)  
[end\\_try\\_branch\(\) \(robot.model.statistics.StatisticsBuilder method\), 236](#)  
[end\\_try\\_branch\(\) \(robot.model.tagsetter.TagSetter method\), 244](#)  
[end\\_try\\_branch\(\) \(robot.model.totalstatistics.TotalStatisticsBuilder method\), 255](#)  
[end\\_try\\_branch\(\) \(robot.model.visitor.SuiteVisitor method\), 261](#)  
[end\\_try\\_branch\(\) \(robot.output.console.dotted.StatusReporter method\), 265](#)  
[end\\_try\\_branch\(\) \(robot.output.xmllogger.XmlLogger method\), 279](#)  
[end\\_try\\_branch\(\) \(robot.reporting.outputwriter.OutputWriter method\), 361](#)  
[end\\_try\\_branch\(\) \(robot.reporting.xunitwriter.XUnitFileWriter method\), 361](#)  
[end\\_try\\_branch\(\) \(robot.result.configurer.SuiteConfigurer method\), 373](#)  
[end\\_try\\_branch\(\) \(robot.result.keywordremover.AllKeywordsRemover method\), 381](#)  
[end\\_try\\_branch\(\) \(robot.result.keywordremover.ByTagKeywordRemover method\), 389](#)  
[end\\_try\\_branch\(\) \(robot.result.keywordremover.ForLoopItemsRemover method\), 397](#)  
[end\\_try\\_branch\(\) \(robot.result.keywordremover.PassedKeywordRemover method\), 385](#)  
[end\\_try\\_branch\(\) \(robot.result.keywordremover.WaitUntilKeywordSucceeded method\), 405](#)  
[end\\_try\\_branch\(\) \(robot.result.keywordremover.WarningAndErrorFinder method\), 410](#)  
[end\\_try\\_branch\(\) \(robot.result.keywordremover.WhileLoopItemsRemover method\), 401](#)  
[end\\_try\\_branch\(\) \(robot.result.merger.Merger method\), 414](#)  
[end\\_try\\_branch\(\) \(robot.result.messagefilter.MessageFilter method\), 418](#)  
[end\\_try\\_branch\(\) \(robot.result.resultbuilder.RemoveKeywords method\), 466](#)  
[end\\_try\\_branch\(\) \(robot.result.suiteardownfailed.SuiteTeardownFailed method\), 474](#)  
[end\\_try\\_branch\(\) \(robot.result.suiteardownfailed.SuiteTeardownFailureHandler method\), 470](#)  
[end\\_try\\_branch\(\) \(robot.result.visitor.ResultVisitor method\), 479](#)  
[end\\_try\\_branch\(\) \(robot.running.randomizer.Randomizer method\), 542](#)  
[end\\_try\\_branch\(\) \(robot.running.suiterunner.SuiteRunner method\), 548](#)  
[end\\_user\\_keyword\(\) \(robot.running.namespace.Namespace method\), 540](#)  
[end\\_while\(\) \(robot.conf.gatherfailed.GatherFailedSuites method\), 28](#)  
[end\\_while\(\) \(robot.conf.gatherfailed.GatherFailedTests method\), 24](#)  
[end\\_while\(\) \(robot.model.configurer.SuiteConfigurer method\), 200](#)  
[end\\_while\(\) \(robot.model.filter.EmptySuiteRemover method\), 216](#)  
[end\\_while\(\) \(robot.model.filter.Filter method\), 220](#)  
[end\\_while\(\) \(robot.model.modifier.ModelModifier method\), 231](#)  
[end\\_while\(\) \(robot.model.statistics.StatisticsBuilder method\), 236](#)  
[end\\_while\(\) \(robot.model.tagsetter.TagSetter method\), 244](#)  
[end\\_while\(\) \(robot.model.totalstatistics.TotalStatisticsBuilder method\), 255](#)

```

        method), 255
end_while() (robot.model.visitor.SuiteVisitor
        method), 261
end_while() (robot.output.console.dotted.StatusReporter
        method), 265
end_while() (robot.output.xmllogger.XmlLogger
        method), 280
end_while() (robot.reporting.outputwriter.OutputWriter
        method), 361
end_while() (robot.reporting.xunitwriter.XUnitFileWriter
        method), 368
end_while() (robot.result.configurer.SuiteConfigurer
        method), 373
end_while() (robot.result.keywordremover.AllKeywordsRemover
        method), 381
end_while() (robot.result.keywordremover.ByNameKeywordRemover
        method), 389
end_while() (robot.result.keywordremover.ByTagKeywordRemover
        method), 393
end_while() (robot.result.keywordremover.ForLoopItemsRemover
        method), 397
end_while() (robot.result.keywordremover.PassedKeywordRemover
        method), 385
end_while() (robot.result.keywordremover.WaitUntilKeywordSucceedsRemover
        method), 406
end_while() (robot.result.keywordremover.WarningAndErrorFinder
        method), 410
end_while() (robot.result.keywordremover.WhileLoopItemsRemover
        method), 401
end_while() (robot.result.merger.Merger
        method), 414
end_while() (robot.result.messagefilter.MessageFilter
        method), 418
end_while() (robot.result.resultbuilder.RemoveKeywords
        method), 466
end_while() (robot.result.suiteteardownfailed.SuiteTeardownFailed
        method), 474
end_while() (robot.result.suiteteardownfailed.SuiteTeardownFailureHandler
        method), 470
end_while() (robot.result.visitor.ResultVisitor
        method), 479
end_while() (robot.running.randomizer.Randomizer
        method), 542
end_while() (robot.running.suiterunner.SuiteRunner
        method), 548
end_while_iteration()
        (robot.conf.gatherfailed.GatherFailedSuites
        method), 28
end_while_iteration()
        (robot.conf.gatherfailed.GatherFailedTests
        method), 24
end_while_iteration()
        (robot.model.configurer.SuiteConfigurer
        method), 200
        end_while_iteration()
        (robot.model.filter.EmptySuiteRemover
        method), 216
        end_while_iteration() (robot.model.filter.Filter
        method), 220
        end_while_iteration()
        (robot.model.modifier.ModelModifier
        method), 232
        end_while_iteration()
        (robot.model.statistics.StatisticsBuilder
        method), 237
        end_while_iteration()
        (robot.model.tagsetter.TagSetter
        method), 244
        end_while_iteration()
        (robot.model.totalstatistics.TotalStatisticsBuilder
        method), 255
        end_while_iteration()
        (robot.model.visitor.SuiteVisitor
        method), 262
        end_while_iteration()
        (robot.output.console.dotted.StatusReporter
        method), 265
        end_while_iteration()
        (robot.output.xmllogger.XmlLogger
        method), 280
        end_while_iteration()
        (robot.reporting.outputwriter.OutputWriter
        method), 361
        end_while_iteration()
        (robot.reporting.xunitwriter.XUnitFileWriter
        method), 368
        end_while_iteration()
        (robot.result.configurer.SuiteConfigurer
        method), 373
        end_while_iteration()
        (robot.result.keywordremover.AllKeywordsRemover
        method), 381
        end_while_iteration()
        (robot.result.keywordremover.ByNameKeywordRemover
        method), 389
        end_while_iteration()
        (robot.result.keywordremover.ByTagKeywordRemover
        method), 393
        end_while_iteration()
        (robot.result.keywordremover.ForLoopItemsRemover
        method), 397
        end_while_iteration()
        (robot.result.keywordremover.PassedKeywordRemover
        method), 385
        end_while_iteration()
        (robot.result.keywordremover.WaitUntilKeywordSucceedsRemover
        method), 406
        end_while_iteration()

```



(robot.result.keywordremover.WarningAndErrorFinder method), 410  
 end\_while\_iteration() (robot.result.keywordremover.WhileLoopItemsRemover method), 401  
 end\_while\_iteration() (robot.result.merger.Merger method), 414  
 end\_while\_iteration() (robot.result.messagefilter.MessageFilter method), 418  
 end\_while\_iteration() (robot.result.resultbuilder.RemoveKeywords method), 466  
 end\_while\_iteration() (robot.result.suiteteardownfailed.SuiteTeardownFailed method), 474  
 end\_while\_iteration() (robot.result.suiteteardownfailed.SuiteTeardownFailedHelper method), 470  
 end\_while\_iteration() (robot.result.visitor.ResultVisitor method), 479  
 end\_while\_iteration() (robot.running.randomizer.Randomizer method), 542  
 end\_while\_iteration() (robot.running.suiterunner.SuiteRunner method), 548  
 EndKeywordArguments (class in robot.output.listenerarguments), 271  
 EndLexer (class in robot.parsing.lexer.statementlexers), 298  
 EndSuiteArguments (class in robot.output.listenerarguments), 271  
 EndTestArguments (class in robot.output.listenerarguments), 271  
 endtime (robot.result.model.Break attribute), 452  
 endtime (robot.result.model.Continue attribute), 449  
 endtime (robot.result.model.For attribute), 431  
 endtime (robot.result.model.ForIteration attribute), 429  
 endtime (robot.result.model.If attribute), 440  
 endtime (robot.result.model.IfBranch attribute), 438  
 endtime (robot.result.model.Keyword attribute), 454  
 endtime (robot.result.model.Return attribute), 447  
 endtime (robot.result.model.TestCase attribute), 457  
 endtime (robot.result.model.TestSuite attribute), 460  
 endtime (robot.result.model.Try attribute), 445  
 endtime (robot.result.model.TryBranch attribute), 442  
 endtime (robot.result.model.While attribute), 436  
 endtime (robot.result.model.WhileIteration attribute), 433  
 ENUM (robot.libdocpkg.datatypes.TypeDoc attribute), 36  
 EnumConverter (class in robot.running.arguments.typeconverters), 495  
 EnumMember (class in robot.libdocpkg.datatypes), 36  
 environment\_variable\_should\_be\_set() (robot.libraries.OperatingSystem.OperatingSystem method), 84  
 environment\_variable\_should\_not\_be\_set() (robot.libraries.OperatingSystem.OperatingSystem method), 85  
 EnvironmentFinder (class in robot.variables.finders), 569  
 EOL (robot.parsing.lexer.tokens.END attribute), 304  
 EOL (robot.parsing.lexer.tokens.EOS attribute), 302  
 EOL (robot.parsing.lexer.tokens.Token attribute), 301  
 Eof (class in robot.parsing.lexer.tokens), 301  
 EOS (robot.parsing.lexer.tokens.END attribute), 304  
 EOS (robot.parsing.lexer.tokens.EOS attribute), 302  
 EOS (robot.parsing.lexer.tokens.Token attribute), 301  
 eq() (in module robot.utils.match), 565  
 Error, 13  
 Error (class in robot.parsing.model.statements), 351  
 ERROR (robot.parsing.lexer.tokens.END attribute), 304  
 error (robot.parsing.lexer.tokens.END attribute), 305  
 ERROR (robot.parsing.lexer.tokens.EOS attribute), 302  
 error (robot.parsing.lexer.tokens.EOS attribute), 303  
 ERROR (robot.parsing.lexer.tokens.Token attribute), 301  
 error (robot.parsing.lexer.tokens.Token attribute), 301  
 error (robot.running.model.Break attribute), 531  
 error (robot.running.model.Continue attribute), 530  
 error (robot.running.model.For attribute), 520  
 error (robot.running.model.If attribute), 524  
 error (robot.running.model.Return attribute), 528  
 error (robot.running.model.Try attribute), 527  
 error (robot.running.model.While attribute), 521  
 error() (in module robot.api.logger), 15  
 error() (in module robot.output.librarylogger), 271  
 error() (robot.output.console.highlighting.HighlightingStream method), 268  
 error() (robot.output.console.verbose.VerboseWriter method), 269  
 error() (robot.output.filelogger.FileLogger method), 270  
 error() (robot.output.logger.Logger method), 274  
 error() (robot.output.loggerhelper.AbstractLogger method), 274  
 error() (robot.output.output.Output method), 276  
 error() (robot.utils.application.DefaultLogger method), 554  
 error() (robot.utils.importer.NoLogger method), 564  
 error\_occurred() (robot.running.status.Exit method), 545  
 error\_occurred() (robot.running.status.SuiteStatus method), 545

- `error_occurred()` (*robot.running.status.TestStatus* method), 545
- `ErrorDetails` (class in *robot.utils.error*), 560
- `ErrorMessageBuilder` (class in *robot.reporting.jsmodelbuilders*), 358
- `ErrorMessageHandler` (class in *robot.result.xmlelementhandlers*), 490
- `ErrorReporter` (class in *robot.running.builder.parsers*), 505
- `errors` (*robot.parsing.model.blocks.Block* attribute), 306
- `errors` (*robot.parsing.model.blocks.CommentSection* attribute), 308
- `errors` (*robot.parsing.model.blocks.File* attribute), 306
- `errors` (*robot.parsing.model.blocks.For* attribute), 309
- `errors` (*robot.parsing.model.blocks.HeaderAndBody* attribute), 306
- `errors` (*robot.parsing.model.blocks.If* attribute), 309
- `errors` (*robot.parsing.model.blocks.Keyword* attribute), 308
- `errors` (*robot.parsing.model.blocks.KeywordSection* attribute), 308
- `errors` (*robot.parsing.model.blocks.Section* attribute), 307
- `errors` (*robot.parsing.model.blocks.SettingSection* attribute), 307
- `errors` (*robot.parsing.model.blocks.TestCase* attribute), 308
- `errors` (*robot.parsing.model.blocks.TestCaseSection* attribute), 307
- `errors` (*robot.parsing.model.blocks.Try* attribute), 309
- `errors` (*robot.parsing.model.blocks.VariableSection* attribute), 307
- `errors` (*robot.parsing.model.blocks.While* attribute), 310
- `errors` (*robot.parsing.model.statements.Error* attribute), 351
- `errors` (*robot.result.executionresult.Result* attribute), 377
- `ErrorsBuilder` (class in *robot.reporting.jsmodelbuilders*), 358
- `ErrorSectionHeaderLexer` (class in *robot.parsing.lexer.statementlexers*), 295
- `ErrorSectionLexer` (class in *robot.parsing.lexer.blocklexers*), 285
- `ErrorsHandler` (class in *robot.result.xmlelementhandlers*), 490
- `escape()` (in module *robot.utils.escaping*), 560
- `ETSource` (class in *robot.utils.etreewrapper*), 560
- `evaluate()` (*robot.libraries.BuiltIn.BuiltIn* method), 45
- `evaluate_expression()` (in module *robot.variables.evaluation*), 568
- `evaluate_xpath()` (*robot.libraries.XML.XML* method), 122
- `EvaluationNamespace` (class in *robot.variables.evaluation*), 568
- `event_add()` (*robot.libraries.dialogs\_py.InputDialog* method), 139
- `event_add()` (*robot.libraries.dialogs\_py.MessageDialog* method), 125
- `event_add()` (*robot.libraries.dialogs\_py.MultipleSelectionDialog* method), 167
- `event_add()` (*robot.libraries.dialogs\_py.PassFailDialog* method), 181
- `event_add()` (*robot.libraries.dialogs\_py.SelectionDialog* method), 153
- `event_delete()` (*robot.libraries.dialogs\_py.InputDialog* method), 139
- `event_delete()` (*robot.libraries.dialogs\_py.MessageDialog* method), 125
- `event_delete()` (*robot.libraries.dialogs\_py.MultipleSelectionDialog* method), 167
- `event_delete()` (*robot.libraries.dialogs\_py.PassFailDialog* method), 181
- `event_delete()` (*robot.libraries.dialogs\_py.SelectionDialog* method), 153
- `event_generate()` (*robot.libraries.dialogs\_py.InputDialog* method), 139
- `event_generate()` (*robot.libraries.dialogs\_py.MessageDialog* method), 125
- `event_generate()` (*robot.libraries.dialogs\_py.MultipleSelectionDialog* method), 167
- `event_generate()` (*robot.libraries.dialogs\_py.PassFailDialog* method), 181
- `event_generate()` (*robot.libraries.dialogs\_py.SelectionDialog* method), 153
- `event_info()` (*robot.libraries.dialogs\_py.InputDialog* method), 139
- `event_info()` (*robot.libraries.dialogs\_py.MessageDialog* method), 125
- `event_info()` (*robot.libraries.dialogs\_py.MultipleSelectionDialog* method), 167
- `event_info()` (*robot.libraries.dialogs\_py.PassFailDialog* method), 181
- `event_info()` (*robot.libraries.dialogs\_py.SelectionDialog* method), 153
- `EXCEPT` (*robot.model.body.BodyItem* attribute), 193
- `EXCEPT` (*robot.model.control.Break* attribute), 214
- `EXCEPT` (*robot.model.control.Continue* attribute), 213
- `EXCEPT` (*robot.model.control.For* attribute), 204
- `EXCEPT` (*robot.model.control.If* attribute), 208
- `EXCEPT` (*robot.model.control.IfBranch* attribute), 206
- `EXCEPT` (*robot.model.control.Return* attribute), 212
- `EXCEPT` (*robot.model.control.Try* attribute), 210
- `EXCEPT` (*robot.model.control.TryBranch* attribute), 209
- `EXCEPT` (*robot.model.control.While* attribute), 205
- `EXCEPT` (*robot.model.keyword.Keyword* attribute), 225

- EXCEPT (*robot.model.message.Message* attribute), 228
- EXCEPT (*robot.output.loggerhelper.Message* attribute), 274
- EXCEPT (*robot.parsing.lexer.tokens.END* attribute), 304
- EXCEPT (*robot.parsing.lexer.tokens.EOS* attribute), 302
- EXCEPT (*robot.parsing.lexer.tokens.Token* attribute), 300
- EXCEPT (*robot.result.model.Break* attribute), 452
- EXCEPT (*robot.result.model.Continue* attribute), 450
- EXCEPT (*robot.result.model.For* attribute), 431
- EXCEPT (*robot.result.model.ForIteration* attribute), 429
- EXCEPT (*robot.result.model.If* attribute), 440
- EXCEPT (*robot.result.model.IfBranch* attribute), 438
- EXCEPT (*robot.result.model.Keyword* attribute), 455
- EXCEPT (*robot.result.model.Message* attribute), 427
- EXCEPT (*robot.result.model.Return* attribute), 447
- EXCEPT (*robot.result.model.Try* attribute), 445
- EXCEPT (*robot.result.model.TryBranch* attribute), 443
- EXCEPT (*robot.result.model.While* attribute), 436
- EXCEPT (*robot.result.model.WhileIteration* attribute), 434
- EXCEPT (*robot.running.model.Break* attribute), 531
- EXCEPT (*robot.running.model.Continue* attribute), 530
- EXCEPT (*robot.running.model.For* attribute), 520
- EXCEPT (*robot.running.model.If* attribute), 524
- EXCEPT (*robot.running.model.IfBranch* attribute), 523
- EXCEPT (*robot.running.model.Keyword* attribute), 518
- EXCEPT (*robot.running.model.Return* attribute), 529
- EXCEPT (*robot.running.model.Try* attribute), 527
- EXCEPT (*robot.running.model.TryBranch* attribute), 526
- EXCEPT (*robot.running.model.While* attribute), 522
- except\_branches (*robot.model.control.Try* attribute), 210
- except\_branches (*robot.result.model.Try* attribute), 446
- except\_branches (*robot.running.model.Try* attribute), 528
- ExceptHeader (class in *robot.parsing.model.statements*), 343
- ExceptHeaderLexer (class in *robot.parsing.lexer.statementslexers*), 297
- exclude (*robot.conf.settings.RebotSettings* attribute), 33
- exclude (*robot.conf.settings.RobotSettings* attribute), 32
- exclude\_tags (*robot.model.filter.Filter* attribute), 219
- execute () (*robot.libdoc.LibDoc* method), 580
- execute () (*robot.rebot.Rebot* method), 582
- execute () (*robot.run.RobotFramework* method), 583
- execute () (*robot.running.timeouts.posix.Timeout* method), 511
- execute () (*robot.running.timeouts.windows.Timeout* method), 511
- execute () (*robot.testdoc.TestDoc* method), 585
- execute () (*robot.utils.application.Application* method), 553
- execute\_cli () (*robot.libdoc.LibDoc* method), 580
- execute\_cli () (*robot.rebot.Rebot* method), 582
- execute\_cli () (*robot.run.RobotFramework* method), 583
- execute\_cli () (*robot.testdoc.TestDoc* method), 585
- execute\_cli () (*robot.utils.application.Application* method), 553
- execute\_command () (*robot.libraries.Telnet.TelnetConnection* method), 109
- execute\_manual\_step () (in module *robot.libraries.Dialogs*), 77
- ExecutionContexts (class in *robot.running.context*), 513
- ExecutionErrors (class in *robot.result.executionerrors*), 376
- ExecutionFailed, 576
- ExecutionFailures, 577
- ExecutionPassed, 577
- ExecutionResult (class in *robot.libraries.Process*), 93
- ExecutionResult () (in module *robot.result.resultbuilder*), 464
- ExecutionResultBuilder (class in *robot.result.resultbuilder*), 464
- ExecutionStatus, 576
- Exit (class in *robot.running.status*), 545
- exit\_for\_loop () (*robot.libraries.BuiltIn.BuiltIn* method), 45
- exit\_for\_loop\_if () (*robot.libraries.BuiltIn.BuiltIn* method), 46
- exit\_on\_error (*robot.conf.settings.RobotSettings* attribute), 31
- exit\_on\_error\_message (*robot.running.status.TestMessage* attribute), 546
- exit\_on\_failure (*robot.conf.settings.RobotSettings* attribute), 31
- exit\_on\_failure\_message (*robot.running.status.TestMessage* attribute), 546
- exit\_on\_fatal\_message (*robot.running.status.TestMessage* attribute), 546
- expand\_keywords (*robot.conf.settings.RebotSettings* attribute), 33
- expand\_keywords (*robot.reporting.jsbuildingcontext.JsBuildingContext* attribute), 357
- ExpandKeywordMatcher (class in *robot.reporting.expandkeywordmatcher*), 357

`expect()` (*robot.libraries.Telnet.TelnetConnection* method), 110  
`extend()` (*robot.model.body.BaseBody* method), 195  
`extend()` (*robot.model.body.Body* method), 196  
`extend()` (*robot.model.body.Branches* method), 198  
`extend()` (*robot.model.itemlist.ItemList* method), 224  
`extend()` (*robot.model.keyword.Keywords* method), 227  
`extend()` (*robot.model.message.Messages* method), 229  
`extend()` (*robot.model.testcase.TestCases* method), 249  
`extend()` (*robot.model.testsuite.TestSuites* method), 253  
`extend()` (*robot.result.model.Body* method), 422  
`extend()` (*robot.result.model.Branches* method), 424  
`extend()` (*robot.result.model.Iterations* method), 425  
`extend()` (*robot.running.model.Body* method), 516  
`extend()` (*robot.running.model.Imports* method), 539  
`ExtendedFinder` (class in *robot.variables.finders*), 569  
`extension` (*robot.conf.settings.RobotSettings* attribute), 32

## F

`FAIL` (*robot.result.model.Break* attribute), 452  
`FAIL` (*robot.result.model.Continue* attribute), 450  
`FAIL` (*robot.result.model.For* attribute), 431  
`FAIL` (*robot.result.model.ForIteration* attribute), 429  
`FAIL` (*robot.result.model.If* attribute), 440  
`FAIL` (*robot.result.model.IfBranch* attribute), 438  
`FAIL` (*robot.result.model.Keyword* attribute), 455  
`FAIL` (*robot.result.model.Return* attribute), 447  
`FAIL` (*robot.result.model.StatusMixin* attribute), 428  
`FAIL` (*robot.result.model.TestCase* attribute), 457  
`FAIL` (*robot.result.model.TestSuite* attribute), 460  
`FAIL` (*robot.result.model.Try* attribute), 445  
`FAIL` (*robot.result.model.TryBranch* attribute), 443  
`FAIL` (*robot.result.model.While* attribute), 436  
`FAIL` (*robot.result.model.WhileIteration* attribute), 434  
`fail()` (in module *robot.utils.asserts*), 556  
`fail()` (*robot.libraries.BuiltIn.BuiltIn* method), 46  
`fail()` (*robot.output.filelogger.FileLogger* method), 270  
`fail()` (*robot.output.logger.Logger* method), 274  
`fail()` (*robot.output.loggerhelper.AbstractLogger* method), 274  
`fail()` (*robot.output.output.Output* method), 276  
`failed` (*robot.model.stats.Stat* attribute), 240  
`failed` (*robot.model.totalstatistics.TotalStatistics* attribute), 253  
`failed` (*robot.result.model.Break* attribute), 453  
`failed` (*robot.result.model.Continue* attribute), 451  
`failed` (*robot.result.model.For* attribute), 432

`failed` (*robot.result.model.ForIteration* attribute), 430  
`failed` (*robot.result.model.If* attribute), 441  
`failed` (*robot.result.model.IfBranch* attribute), 439  
`failed` (*robot.result.model.Keyword* attribute), 456  
`failed` (*robot.result.model.Return* attribute), 448  
`failed` (*robot.result.model.StatusMixin* attribute), 428  
`failed` (*robot.result.model.TestCase* attribute), 458  
`failed` (*robot.result.model.TestSuite* attribute), 460  
`failed` (*robot.result.model.Try* attribute), 446  
`failed` (*robot.result.model.TryBranch* attribute), 444  
`failed` (*robot.result.model.While* attribute), 437  
`failed` (*robot.result.model.WhileIteration* attribute), 435  
`failed` (*robot.running.status.SuiteStatus* attribute), 545  
`failed` (*robot.running.status.TestStatus* attribute), 546  
`Failure`, 12  
`Failure` (class in *robot.running.status*), 545  
`failure_occurred()` (*robot.running.status.Exit* method), 545  
`failure_occurred()` (*robot.running.status.SuiteStatus* method), 545  
`failure_occurred()` (*robot.running.status.TestStatus* method), 546  
`FATAL_ERROR` (*robot.parsing.lexer.tokens.END* attribute), 304  
`FATAL_ERROR` (*robot.parsing.lexer.tokens.EOS* attribute), 302  
`FATAL_ERROR` (*robot.parsing.lexer.tokens.Token* attribute), 301  
`fatal_error()` (*robot.libraries.BuiltIn.BuiltIn* method), 46  
`FatalError`, 13  
`feed()` (*robot.libraries.Telnet.TerminalEmulator* method), 111  
`fetch_from_left()` (*robot.libraries.String.String* method), 101  
`fetch_from_right()` (*robot.libraries.String.String* method), 101  
`File` (class in *robot.parsing.model.blocks*), 306  
`file_should_be_empty()` (*robot.libraries.OperatingSystem.OperatingSystem* method), 82  
`file_should_exist()` (*robot.libraries.OperatingSystem.OperatingSystem* method), 81  
`file_should_not_be_empty()` (*robot.libraries.OperatingSystem.OperatingSystem* method), 82  
`file_should_not_exist()` (*robot.libraries.OperatingSystem.OperatingSystem* method), 81  
`FileContext` (class in *robot.parsing.lexer.context*),



- 288
- FileLexer (class in robot.parsing.lexer.blocklexers), 283
- FileLogger (class in robot.output.filelogger), 270
- fileno() (robot.libraries.Telnet.TelnetConnection method), 110
- FileParser (class in robot.parsing.parser.fileparser), 354
- FileReader (class in robot.utils.filereader), 560
- fill\_named() (robot.running.arguments.argumentmapper.KeywordCallTemplate method), 492
- fill\_positional() (robot.running.arguments.argumentmapper.KeywordCallTemplate method), 492
- fill\_rawq() (robot.libraries.Telnet.TelnetConnection method), 110
- Filter (class in robot.model.filter), 219
- filter() (robot.model.body.BaseBody method), 195
- filter() (robot.model.body.Body method), 196
- filter() (robot.model.body.Branches method), 198
- filter() (robot.model.testsuite.TestSuite method), 251
- filter() (robot.output.pyloggingconf.RobotHandler method), 277
- filter() (robot.result.model.Body method), 422
- filter() (robot.result.model.Branches method), 424
- filter() (robot.result.model.Iterations method), 426
- filter() (robot.result.model.TestSuite method), 460
- filter() (robot.running.model.Body method), 516
- filter() (robot.running.model.TestSuite method), 536
- filter\_messages() (robot.result.model.TestSuite method), 463
- FINALLY (robot.model.body.BodyItem attribute), 193
- FINALLY (robot.model.control.Break attribute), 214
- FINALLY (robot.model.control.Continue attribute), 213
- FINALLY (robot.model.control.For attribute), 204
- FINALLY (robot.model.control.If attribute), 208
- FINALLY (robot.model.control.IfBranch attribute), 206
- FINALLY (robot.model.control.Return attribute), 212
- FINALLY (robot.model.control.Try attribute), 211
- FINALLY (robot.model.control.TryBranch attribute), 209
- FINALLY (robot.model.control.While attribute), 205
- FINALLY (robot.model.keyword.Keyword attribute), 225
- FINALLY (robot.model.message.Message attribute), 228
- FINALLY (robot.output.loggerhelper.Message attribute), 275
- FINALLY (robot.parsing.lexer.tokens.END attribute), 304
- FINALLY (robot.parsing.lexer.tokens.EOS attribute), 302
- FINALLY (robot.parsing.lexer.tokens.Token attribute), 300
- FINALLY (robot.result.model.Break attribute), 452
- FINALLY (robot.result.model.Continue attribute), 450
- FINALLY (robot.result.model.For attribute), 431
- FINALLY (robot.result.model.ForIteration attribute), 429
- FINALLY (robot.result.model.If attribute), 440
- FINALLY (robot.result.model.IfBranch attribute), 438
- FINALLY (robot.result.model.Keyword attribute), 455
- FINALLY (robot.result.model.Message attribute), 427
- FINALLY (robot.result.model.Return attribute), 447
- FINALLY (robot.result.model.Try attribute), 445
- FINALLY (robot.result.model.TryBranch attribute), 443
- FINALLY (robot.result.model.While attribute), 436
- FINALLY (robot.result.model.WhileIteration attribute), 434
- FINALLY (robot.running.model.Break attribute), 531
- FINALLY (robot.running.model.Continue attribute), 530
- FINALLY (robot.running.model.For attribute), 520
- FINALLY (robot.running.model.If attribute), 524
- FINALLY (robot.running.model.IfBranch attribute), 523
- FINALLY (robot.running.model.Keyword attribute), 518
- FINALLY (robot.running.model.Return attribute), 529
- FINALLY (robot.running.model.Try attribute), 527
- FINALLY (robot.running.model.TryBranch attribute), 526
- FINALLY (robot.running.model.While attribute), 522
- finally\_branch (robot.model.control.Try attribute), 210
- finally\_branch (robot.result.model.Try attribute), 446
- finally\_branch (robot.running.model.Try attribute), 528
- FinallyHeader (class in robot.parsing.model.statements), 344
- FinallyHeaderLexer (class in robot.parsing.lexer.statementlexers), 298
- find() (robot.utils.recommendations.RecommendationFinder method), 567
- find() (robot.variables.finders.EmptyFinder method), 569
- find() (robot.variables.finders.EnvironmentFinder method), 569
- find() (robot.variables.finders.ExtendedFinder method), 569
- find() (robot.variables.finders.InlinePythonFinder method), 569
- find() (robot.variables.finders.NumberFinder method), 569
- find() (robot.variables.finders.StoredFinder method), 569
- find() (robot.variables.finders.VariableFinder method), 569
- find\_all() (robot.libraries.XML.ElementFinder method), 122
- find\_and\_format() (robot.utils.recommendations.RecommendationFinder method), 567

- method*), 567
- find\_from()* (*robot.parsing.model.blocks.FirstStatementFinder* class method), 311
- find\_from()* (*robot.parsing.model.blocks.LastStatementFinder* class method), 311
- FirstStatementFinder* (class in *robot.parsing.model.blocks*), 310
- Fixture* (class in *robot.parsing.model.statements*), 314
- fixture\_class* (*robot.model.testcase.TestCase* attribute), 247
- fixture\_class* (*robot.model.testsuite.TestSuite* attribute), 250
- fixture\_class* (*robot.result.model.TestCase* attribute), 457
- fixture\_class* (*robot.result.model.TestSuite* attribute), 460
- fixture\_class* (*robot.running.model.TestCase* attribute), 532
- fixture\_class* (*robot.running.model.TestSuite* attribute), 534
- flatten()* (*robot.model.body.BaseBody* method), 195
- flatten()* (*robot.model.body.Body* method), 196
- flatten()* (*robot.model.body.Branches* method), 198
- flatten()* (*robot.result.model.Body* method), 422
- flatten()* (*robot.result.model.Branches* method), 424
- flatten()* (*robot.result.model.Iterations* method), 426
- flatten()* (*robot.running.model.Body* method), 517
- flatten\_keywords* (*robot.conf.settings.RebotSettings* attribute), 33
- flatten\_keywords* (*robot.conf.settings.RobotSettings* attribute), 32
- FlattenByNameMatcher* (class in *robot.result.flattenkeywordmatcher*), 379
- FlattenByTagMatcher* (class in *robot.result.flattenkeywordmatcher*), 379
- FlattenByTypeMatcher* (class in *robot.result.flattenkeywordmatcher*), 379
- flavor* (*robot.model.control.For* attribute), 203
- flavor* (*robot.parsing.model.blocks.For* attribute), 309
- flavor* (*robot.parsing.model.statements.ForHeader* attribute), 337
- flavor* (*robot.result.model.For* attribute), 432
- flavor* (*robot.running.bodyrunner.ForInEnumerateRunner* attribute), 511
- flavor* (*robot.running.bodyrunner.ForInRangeRunner* attribute), 511
- flavor* (*robot.running.bodyrunner.ForInRunner* attribute), 511
- flavor* (*robot.running.bodyrunner.ForInZipRunner* attribute), 511
- flavor* (*robot.running.model.For* attribute), 521
- FloatConverter* (class in *robot.running.arguments.typeconverters*), 497
- flush()* (*robot.output.console.highlighting.HighlightingStream* method), 268
- flush()* (*robot.output.pyloggingconf.RobotHandler* method), 277
- focus()* (*robot.libraries.dialogs\_py.InputDialog* method), 139
- focus()* (*robot.libraries.dialogs\_py.MessageDialog* method), 125
- focus()* (*robot.libraries.dialogs\_py.MultipleSelectionDialog* method), 167
- focus()* (*robot.libraries.dialogs\_py.PassFailDialog* method), 181
- focus()* (*robot.libraries.dialogs\_py.SelectionDialog* method), 153
- focus\_displayof()* (*robot.libraries.dialogs\_py.InputDialog* method), 139
- focus\_displayof()* (*robot.libraries.dialogs\_py.MessageDialog* method), 126
- focus\_displayof()* (*robot.libraries.dialogs\_py.MultipleSelectionDialog* method), 167
- focus\_displayof()* (*robot.libraries.dialogs\_py.PassFailDialog* method), 181
- focus\_displayof()* (*robot.libraries.dialogs\_py.SelectionDialog* method), 153
- focus\_force()* (*robot.libraries.dialogs\_py.InputDialog* method), 140
- focus\_force()* (*robot.libraries.dialogs\_py.MessageDialog* method), 126
- focus\_force()* (*robot.libraries.dialogs\_py.MultipleSelectionDialog* method), 168
- focus\_force()* (*robot.libraries.dialogs\_py.PassFailDialog* method), 182
- focus\_force()* (*robot.libraries.dialogs\_py.SelectionDialog* method), 154
- focus\_get()* (*robot.libraries.dialogs\_py.InputDialog* method), 140
- focus\_get()* (*robot.libraries.dialogs\_py.MessageDialog* method), 126
- focus\_get()* (*robot.libraries.dialogs\_py.MultipleSelectionDialog* method), 168
- focus\_get()* (*robot.libraries.dialogs\_py.PassFailDialog* method), 182
- focus\_get()* (*robot.libraries.dialogs\_py.SelectionDialog* method), 154
- focus\_lastfor()* (*robot.libraries.dialogs\_py.InputDialog* method), 140
- focus\_lastfor()* (*robot.libraries.dialogs\_py.MessageDialog* method), 126
- focus\_lastfor()* (*robot.libraries.dialogs\_py.MultipleSelectionDialog* method), 168

- method*), 168
- `focus_lastfor()` (*robot.libraries.dialogs\_py.PassFailDialog method*), 182
- `focus_lastfor()` (*robot.libraries.dialogs\_py.SelectionDialog method*), 154
- `focus_set()` (*robot.libraries.dialogs\_py.InputDialog method*), 140
- `focus_set()` (*robot.libraries.dialogs\_py.MessageDialog method*), 126
- `focus_set()` (*robot.libraries.dialogs\_py.MultipleSelectionDialog method*), 168
- `focus_set()` (*robot.libraries.dialogs\_py.PassFailDialog method*), 182
- `focus_set()` (*robot.libraries.dialogs\_py.SelectionDialog method*), 154
- `focusmodel()` (*robot.libraries.dialogs\_py.InputDialog method*), 140
- `focusmodel()` (*robot.libraries.dialogs\_py.MessageDialog method*), 126
- `focusmodel()` (*robot.libraries.dialogs\_py.MultipleSelectionDialog method*), 168
- `focusmodel()` (*robot.libraries.dialogs\_py.PassFailDialog method*), 182
- `focusmodel()` (*robot.libraries.dialogs\_py.SelectionDialog method*), 154
- `For` (*class in robot.model.control*), 203
- `For` (*class in robot.parsing.model.blocks*), 309
- `For` (*class in robot.result.model*), 431
- `For` (*class in robot.running.model*), 520
- `FOR` (*robot.model.body.BodyItem attribute*), 193
- `FOR` (*robot.model.control.Break attribute*), 214
- `FOR` (*robot.model.control.Continue attribute*), 213
- `FOR` (*robot.model.control.For attribute*), 204
- `FOR` (*robot.model.control.If attribute*), 208
- `FOR` (*robot.model.control.IfBranch attribute*), 206
- `FOR` (*robot.model.control.Return attribute*), 212
- `FOR` (*robot.model.control.Try attribute*), 211
- `FOR` (*robot.model.control.TryBranch attribute*), 209
- `FOR` (*robot.model.control.While attribute*), 205
- `FOR` (*robot.model.keyword.Keyword attribute*), 225
- `FOR` (*robot.model.message.Message attribute*), 228
- `FOR` (*robot.output.loggerhelper.Message attribute*), 275
- `FOR` (*robot.parsing.lexer.tokens.END attribute*), 304
- `FOR` (*robot.parsing.lexer.tokens.EOS attribute*), 302
- `FOR` (*robot.parsing.lexer.tokens.Token attribute*), 300
- `FOR` (*robot.result.model.Break attribute*), 452
- `FOR` (*robot.result.model.Continue attribute*), 450
- `FOR` (*robot.result.model.For attribute*), 431
- `FOR` (*robot.result.model.ForIteration attribute*), 429
- `FOR` (*robot.result.model.If attribute*), 440
- `FOR` (*robot.result.model.IfBranch attribute*), 438
- `FOR` (*robot.result.model.Keyword attribute*), 455
- `FOR` (*robot.result.model.Message attribute*), 427
- `FOR` (*robot.result.model.Return attribute*), 447
- `FOR` (*robot.result.model.Try attribute*), 445
- `FOR` (*robot.result.model.TryBranch attribute*), 443
- `FOR` (*robot.result.model.While attribute*), 436
- `FOR` (*robot.result.model.WhileIteration attribute*), 434
- `FOR` (*robot.running.model.Break attribute*), 531
- `FOR` (*robot.running.model.Continue attribute*), 530
- `FOR` (*robot.running.model.For attribute*), 520
- `FOR` (*robot.running.model.If attribute*), 524
- `FOR` (*robot.running.model.IfBranch attribute*), 523
- `FOR` (*robot.running.model.Keyword attribute*), 518
- `FOR` (*robot.running.model.Return attribute*), 529
- `FOR` (*robot.running.model.Try attribute*), 527
- `FOR` (*robot.running.model.TryBranch attribute*), 526
- `FOR` (*robot.running.model.While attribute*), 522
- `for_class` (*robot.model.body.BaseBody attribute*), 194
- `for_class` (*robot.model.body.Body attribute*), 197
- `for_class` (*robot.model.body.Branches attribute*), 198
- `for_class` (*robot.result.model.Body attribute*), 423
- `for_class` (*robot.result.model.Branches attribute*), 424
- `for_class` (*robot.result.model.Iterations attribute*), 426
- `for_class` (*robot.running.model.Body attribute*), 517
- `for_converter()` (*robot.running.arguments.customconverters.Convert class method*), 495
- `for_enum()` (*robot.libdocpkg.datatypes.TypeDoc class method*), 36
- `FOR_SEPARATOR` (*robot.parsing.lexer.tokens.END attribute*), 304
- `FOR_SEPARATOR` (*robot.parsing.lexer.tokens.EOS attribute*), 302
- `FOR_SEPARATOR` (*robot.parsing.lexer.tokens.Token attribute*), 300
- `for_type()` (*robot.libdocpkg.datatypes.TypeDoc class method*), 36
- `for_typed_dict()` (*robot.libdocpkg.datatypes.TypeDoc class method*), 36
- `ForBuilder` (*class in robot.running.builder.transformers*), 508
- `FORCE_TAGS` (*robot.parsing.lexer.tokens.END attribute*), 304
- `FORCE_TAGS` (*robot.parsing.lexer.tokens.EOS attribute*), 302
- `FORCE_TAGS` (*robot.parsing.lexer.tokens.Token attribute*), 300
- `force_tags` (*robot.running.builder.testsettings.TestDefaults attribute*), 506
- `ForceTags` (*class in robot.parsing.model.statements*), 320
- `forget()` (*robot.libraries.dialogs\_py.InputDialog method*), 140
- `forget()` (*robot.libraries.dialogs\_py.MessageDialog method*), 126

`forget()` (`robot.libraries.dialogs_py.MultipleSelectionDialog` method), 168

`forget()` (`robot.libraries.dialogs_py.PassFailDialog` method), 182

`forget()` (`robot.libraries.dialogs_py.SelectionDialog` method), 154

`ForHandler` (class in `robot.result.xmlelementhandlers`), 484

`ForHeader` (class in `robot.parsing.model.statements`), 337

`ForHeaderLexer` (class in `robot.parsing.lexer.statemntlexers`), 296

`ForInEnumerateRunner` (class in `robot.running.bodyrunner`), 511

`ForInRangeRunner` (class in `robot.running.bodyrunner`), 511

`ForInRunner` (class in `robot.running.bodyrunner`), 511

`ForInZipRunner` (class in `robot.running.bodyrunner`), 511

`ForIteration` (class in `robot.result.model`), 428

`ForLexer` (class in `robot.parsing.lexer.blocklexers`), 287

`ForLoopItemsRemover` (class in `robot.result.keywordremover`), 396

`format()` (`robot.output.pyloggingconf.RobotHandler` method), 277

`format()` (`robot.utils.htmlformatters.HeaderFormatter` method), 562

`format()` (`robot.utils.htmlformatters.HtmlFormatter` method), 561

`format()` (`robot.utils.htmlformatters.LineFormatter` method), 561

`format()` (`robot.utils.htmlformatters.ListFormatter` method), 562

`format()` (`robot.utils.htmlformatters.ParagraphFormatter` method), 562

`format()` (`robot.utils.htmlformatters.PreformattedFormatter` method), 562

`format()` (`robot.utils.htmlformatters.RulerFormatter` method), 561

`format()` (`robot.utils.htmlformatters.TableFormatter` method), 562

`format()` (`robot.utils.recommendations.RecommendationFinder` method), 567

`format_error()` (in module `robot.running.builder.transformers`), 510

`format_line()` (`robot.utils.htmlformatters.HeaderFormatter` method), 562

`format_line()` (`robot.utils.htmlformatters.RulerFormatter` method), 561

`format_link()` (`robot.utils.htmlformatters.LinkFormatter` method), 561

`format_name()` (in module `robot.running.builder.parsers`), 505

`format_recommendations()` (`robot.running.namespace.KeywordRecommendationFinder` static method), 540

`format_string()` (`robot.libraries.String.String` method), 98

`format_url()` (`robot.utils.htmlformatters.LinkFormatter` method), 561

`ForParser` (class in `robot.parsing.parser.blockparsers`), 354

`ForRunner()` (in module `robot.running.bodyrunner`), 511

`frame()` (`robot.libraries.dialogs_py.InputDialog` method), 140

`frame()` (`robot.libraries.dialogs_py.MessageDialog` method), 126

`frame()` (`robot.libraries.dialogs_py.MultipleSelectionDialog` method), 168

`frame()` (`robot.libraries.dialogs_py.PassFailDialog` method), 182

`frame()` (`robot.libraries.dialogs_py.SelectionDialog` method), 154

`FrameworkError`, 574

`frange()` (in module `robot.utils.frange`), 561

`from_bytes()` (`robot.reporting.stringcache.StringIndex` method), 365

`from_dict()` (`robot.running.arguments.customconverters.CustomArgument` class method), 495

`from_file_system()` (`robot.running.model.TestSuite` class method), 534

`from_model()` (`robot.running.model.TestSuite` class method), 535

`from_params()` (`robot.parsing.model.statements.Arguments` class method), 333

`from_params()` (`robot.parsing.model.statements.Break` class method), 349

`from_params()` (`robot.parsing.model.statements.Comment` class method), 350

`from_params()` (`robot.parsing.model.statements.Continue` class method), 349

`from_params()` (`robot.parsing.model.statements.DefaultTags` class method), 321

`from_params()` (`robot.parsing.model.statements.Documentation` class method), 318

`from_params()` (`robot.parsing.model.statements.DocumentationOrMeta` class method), 312

`from_params()` (`robot.parsing.model.statements.ElseHeader` class method), 341

`from_params()` (`robot.parsing.model.statements.ElseIfHeader` class method), 340

`from_params()` (`robot.parsing.model.statements.EmptyLine` class method), 352

`from_params()` (`robot.parsing.model.statements.End`



*class method*), 345  
*from\_params()* (*robot.parsing.model.statements.Error* *class method*), 351  
*from\_params()* (*robot.parsing.model.statements.ExceptHeader* *class method*), 343  
*from\_params()* (*robot.parsing.model.statements.FinallyHeader* *class method*), 344  
*from\_params()* (*robot.parsing.model.statements.Fixture* *class method*), 314  
*from\_params()* (*robot.parsing.model.statements.ForceTags* *class method*), 320  
*from\_params()* (*robot.parsing.model.statements.ForHeader* *class method*), 337  
*from\_params()* (*robot.parsing.model.statements.IfElseHeader* *class method*), 338  
*from\_params()* (*robot.parsing.model.statements.IfHeader* *class method*), 338  
*from\_params()* (*robot.parsing.model.statements.InlineIfHeader* *class method*), 339  
*from\_params()* (*robot.parsing.model.statements.KeywordCall* *class method*), 335  
*from\_params()* (*robot.parsing.model.statements.KeywordName* *class method*), 328  
*from\_params()* (*robot.parsing.model.statements.LibraryImport* *class method*), 316  
*from\_params()* (*robot.parsing.model.statements.LoopControl* *class method*), 348  
*from\_params()* (*robot.parsing.model.statements.Metadata* *class method*), 319  
*from\_params()* (*robot.parsing.model.statements.MultiValue* *class method*), 313  
*from\_params()* (*robot.parsing.model.statements.NoArgumentHeader* *class method*), 342  
*from\_params()* (*robot.parsing.model.statements.ResourceImport* *class method*), 317  
*from\_params()* (*robot.parsing.model.statements.Return* *class method*), 334  
*from\_params()* (*robot.parsing.model.statements.ReturnStatement* *class method*), 347  
*from\_params()* (*robot.parsing.model.statements.SectionHeader* *class method*), 315  
*from\_params()* (*robot.parsing.model.statements.Setup* *class method*), 329  
*from\_params()* (*robot.parsing.model.statements.SingleValue* *class method*), 313  
*from\_params()* (*robot.parsing.model.statements.Statement* *class method*), 311  
*from\_params()* (*robot.parsing.model.statements.SuiteSetup* *class method*), 322  
*from\_params()* (*robot.parsing.model.statements.SuiteTeardown* *class method*), 323  
*from\_params()* (*robot.parsing.model.statements.Tags* *class method*), 331  
*from\_params()* (*robot.parsing.model.statements.Teardown* *class method*), 330  
*from\_params()* (*robot.parsing.model.statements.Template* *class method*), 332  
*from\_params()* (*robot.parsing.model.statements.TemplateArguments* *class method*), 336  
*from\_params()* (*robot.parsing.model.statements.TestCaseName* *class method*), 328  
*from\_params()* (*robot.parsing.model.statements.TestSetup* *class method*), 323  
*from\_params()* (*robot.parsing.model.statements.TestTeardown* *class method*), 324  
*from\_params()* (*robot.parsing.model.statements.TestTemplate* *class method*), 325  
*from\_params()* (*robot.parsing.model.statements.TestTimeout* *class method*), 326  
*from\_params()* (*robot.parsing.model.statements.Timeout* *class method*), 333  
*from\_params()* (*robot.parsing.model.statements.TryHeader* *class method*), 343  
*from\_params()* (*robot.parsing.model.statements.Variable* *class method*), 327  
*from\_params()* (*robot.parsing.model.statements.VariablesImport* *class method*), 318  
*from\_params()* (*robot.parsing.model.statements.WhileHeader* *class method*), 346  
*from\_token()* (*robot.parsing.lexer.tokens.END* *class method*), 303  
*from\_token()* (*robot.parsing.lexer.tokens.EOS* *class method*), 301  
*from\_tokens()* (*robot.parsing.model.statements.Arguments* *class method*), 334  
*from\_tokens()* (*robot.parsing.model.statements.Break* *class method*), 350  
*from\_tokens()* (*robot.parsing.model.statements.Comment* *class method*), 350  
*from\_tokens()* (*robot.parsing.model.statements.Continue* *class method*), 349  
*from\_tokens()* (*robot.parsing.model.statements.DefaultTags* *class method*), 321  
*from\_tokens()* (*robot.parsing.model.statements.Documentation* *class method*), 319  
*from\_tokens()* (*robot.parsing.model.statements.DocumentationOrMeta* *class method*), 312  
*from\_tokens()* (*robot.parsing.model.statements.ElseHeader* *class method*), 341  
*from\_tokens()* (*robot.parsing.model.statements.ElseIfHeader* *class method*), 340  
*from\_tokens()* (*robot.parsing.model.statements.EmptyLine* *class method*), 352  
*from\_tokens()* (*robot.parsing.model.statements.End* *class method*), 345  
*from\_tokens()* (*robot.parsing.model.statements.Error* *class method*), 351  
*from\_tokens()* (*robot.parsing.model.statements.ExceptHeader*

`from_tokens()` (`robot.parsing.model.statements.FinallyHeader` class method), 344  
`from_tokens()` (`robot.parsing.model.statements.TestCaseName` class method), 345  
`from_tokens()` (`robot.parsing.model.statements.Fixture` class method), 315  
`from_tokens()` (`robot.parsing.model.statements.ForceTags` class method), 320  
`from_tokens()` (`robot.parsing.model.statements.ForHeader` class method), 337  
`from_tokens()` (`robot.parsing.model.statements.IfElseHeader` class method), 338  
`from_tokens()` (`robot.parsing.model.statements.IfHeader` class method), 339  
`from_tokens()` (`robot.parsing.model.statements.InlineIfHeader` class method), 340  
`from_tokens()` (`robot.parsing.model.statements.KeywordCall` class method), 335  
`from_tokens()` (`robot.parsing.model.statements.KeywordName` class method), 329  
`from_tokens()` (`robot.parsing.model.statements.LibraryImport` class method), 316  
`from_tokens()` (`robot.parsing.model.statements.LoopControl` class method), 348  
`from_tokens()` (`robot.parsing.model.statements.Metadata` class method), 320  
`from_tokens()` (`robot.parsing.model.statements.MultiValueListMessage` class method), 314  
`from_tokens()` (`robot.parsing.model.statements.NoArgumentInHeader` class method), 342  
`from_tokens()` (`robot.parsing.model.statements.ResourceImport` class method), 317  
`from_tokens()` (`robot.parsing.model.statements.ReturnStatement` class method), 335  
`from_tokens()` (`robot.parsing.model.statements.ReturnStatement` class method), 347  
`from_tokens()` (`robot.parsing.model.statements.SectionHeader` class method), 315  
`from_tokens()` (`robot.parsing.model.statements.Setup` class method), 330  
`from_tokens()` (`robot.parsing.model.statements.SingleValue` class method), 313  
`from_tokens()` (`robot.parsing.model.statements.StatementGeneratorWriter` class method), 311  
`from_tokens()` (`robot.parsing.model.statements.SuiteSetup` class method), 322  
`from_tokens()` (`robot.parsing.model.statements.SuiteTeardown` class method), 323  
`from_tokens()` (`robot.parsing.model.statements.Tags` class method), 331  
`from_tokens()` (`robot.parsing.model.statements.Teardown` class method), 330  
`from_tokens()` (`robot.parsing.model.statements.Template` class method), 332  
`from_tokens()` (`robot.parsing.model.statements.TemplateArguments`

`gather_failed_suites()` (in module `robot.conf.gatherfailed`), 31  
`gather_failed_tests()` (in module `robot.conf.gatherfailed`), 31  
`GatherFailedSuites` (class in `robot.conf.gatherfailed`), 27  
`GatherFailedTests` (class in `robot.conf.gatherfailed`), 23  
`generate_random_string()` (`robot.libraries.String.String` method), 101  
`GenericVisit` (class in `robot.html.data.htmlfilewriter`), 34  
`generic_visit()` (`robot.parsing.model.blocks.FirstStatementFinder` method), 311  
`generic_visit()` (`robot.parsing.model.blocks.LastStatementFinder` method), 311  
`generic_visit()` (`robot.parsing.model.blocks.ModelValidator` method), 310  
`generic_visit()` (`robot.parsing.model.blocks.ModelWriter` method), 310  
`generic_visit()` (`robot.parsing.model.visitor.ModelTransformer` method), 353

`generic_visit()` (`robot.parsing.model.visitor.ModelVisitor` method), 271  
`method`), 353 `get_attributes()` (`robot.model.stats.CombinedTagStat`  
`generic_visit()` (`robot.running.builder.parsers.ErrorReporter` method), 241  
`method`), 505 `get_attributes()` (`robot.model.stats.Stat` method),  
`generic_visit()` (`robot.running.builder.transformers.ForBuilder` method), 240  
`method`), 508 `get_attributes()` (`robot.model.stats.SuiteStat`  
`generic_visit()` (`robot.running.builder.transformers.IfBuilder` method), 240  
`method`), 509 `get_attributes()` (`robot.model.stats.TagStat`  
`generic_visit()` (`robot.running.builder.transformers.KeywordBuilder` method), 241  
`method`), 508 `get_attributes()` (`robot.model.stats.TotalStat`  
`generic_visit()` (`robot.running.builder.transformers.ResourceBuilder` method), 240  
`method`), 507 `get_binary_file()`  
`generic_visit()` (`robot.running.builder.transformers.SettingsBuilder` method), 506  
`method`), 80 (`robot.libraries.OperatingSystem.OperatingSystem`  
`generic_visit()` (`robot.running.builder.transformers.SuiteBuilder` method), 507  
`method`), 507 `get_child_handler()` (`robot.result.xmlhandlers.ElementHandler`  
`generic_visit()` (`robot.running.builder.transformers.TestCaseBuilder` method), 507  
`method`), 116 `get_child_handler()` (`robot.result.xmlhandlers.ArgumentHandler`  
`generic_visit()` (`robot.running.builder.transformers.TryBuilder` method), 509  
`method`), 490 `get_child_handler()`  
`generic_visit()` (`robot.running.builder.transformers.WhileBuilder` method), 510  
`method`), 489 `get_child_handler()`  
`geometry()` (`robot.libraries.dialogs_py.InputDialog` method), 140  
`method`), 489 `get_child_handler()`  
`geometry()` (`robot.libraries.dialogs_py.MessageDialog` method), 126  
`method`), 489 `get_child_handler()`  
`geometry()` (`robot.libraries.dialogs_py.MultipleSelectionDialog` method), 168  
`method`), 485 `get_child_handler()`  
`geometry()` (`robot.libraries.dialogs_py.PassFailDialog` method), 182  
`method`), 486 `get_child_handler()`  
`geometry()` (`robot.libraries.dialogs_py.SelectionDialog` method), 154  
`method`), 486 `get_child_handler()`  
`get()` (`robot.model.metadata.Metadata` method), 229  
`method`), 559 `get_child_handler()`  
`get()` (`robot.utils.dotdict.DotDict` method), 559  
`method`), 566 `get_child_handler()`  
`get()` (`robot.utils.normalizing.NormalizedDict` method), 566  
`method`), 487 `get_child_handler()`  
`get()` (`robot.variables.evaluation.EvaluationNamespace` method), 568  
`method`), 487 `get_child_handler()`  
`get()` (`robot.variables.store.VariableStore` method), 573  
`method`), 483 `get_child_handler()`  
`get_arguments()` (`robot.output.listenerarguments.EndKeywordArguments` method), 271  
`method`), 483 `get_child_handler()`  
`get_arguments()` (`robot.output.listenerarguments.EndSuiteArguments` method), 271  
`method`), 490 `get_child_handler()`  
`get_arguments()` (`robot.output.listenerarguments.EndTestArguments` method), 271  
`method`), 484 `get_child_handler()`  
`get_arguments()` (`robot.output.listenerarguments.ListenerArguments` method), 271  
`method`), 485 `get_child_handler()`  
`get_arguments()` (`robot.output.listenerarguments.MessageArguments` method), 271  
`method`), 484 `get_child_handler()`  
`get_arguments()` (`robot.output.listenerarguments.StartKeywordArguments` method), 271  
`method`), 485 `get_child_handler()`  
`get_arguments()` (`robot.output.listenerarguments.StartSuiteArguments` method), 271  
`method`), 485 `get_child_handler()`  
`get_arguments()` (`robot.output.listenerarguments.StartTestArguments` method), 271  
`method`), 485

<code>get_child_handler()</code> ( <i>robot.result.xmllementhandlers.IterationHandler</i> <i>method</i> ), 485	<code>get_child_handler()</code> ( <i>robot.result.xmllementhandlers.ValueHandler</i> <i>method</i> ), 490
<code>get_child_handler()</code> ( <i>robot.result.xmllementhandlers.KeywordHandler</i> <i>method</i> ), 484	<code>get_child_handler()</code> ( <i>robot.result.xmllementhandlers.VarHandler</i> <i>method</i> ), 489
<code>get_child_handler()</code> ( <i>robot.result.xmllementhandlers.MessageHandler</i> <i>method</i> ), 487	<code>get_child_handler()</code> ( <i>robot.result.xmllementhandlers.WhileHandler</i> <i>method</i> ), 485
<code>get_child_handler()</code> ( <i>robot.result.xmllementhandlers.MetadataHandler</i> <i>method</i> ), 487	<code>get_combined_stats()</code> ( <i>robot.model.tagstatistics.TagStatInfo</i> <i>method</i> ), 247
<code>get_child_handler()</code> ( <i>robot.result.xmllementhandlers.MetadataItemHandler</i> <i>method</i> ), 488	<code>get_command()</code> ( <i>robot.libraries.Process.ProcessConfiguration</i> <i>method</i> ), 93
<code>get_child_handler()</code> ( <i>robot.result.xmllementhandlers.MetaHandler</i> <i>method</i> ), 488	<code>get_connection()</code> ( <i>robot.utils.connectioncache.ConnectionCache</i> <i>method</i> ), 558
<code>get_child_handler()</code> ( <i>robot.result.xmllementhandlers.PatternHandler</i> <i>method</i> ), 486	<code>get_console_encoding()</code> (in <i>module</i> <i>robot.utils.encodingssniffer</i> ), 560
<code>get_child_handler()</code> ( <i>robot.result.xmllementhandlers.ReturnHandler</i> <i>method</i> ), 486	<code>get_converter_info()</code> ( <i>robot.running.arguments.customconverters.CustomArgumentCor</i> <i>method</i> ), 495
<code>get_child_handler()</code> ( <i>robot.result.xmllementhandlers.RobotHandler</i> <i>method</i> ), 483	<code>get_count()</code> ( <i>robot.libraries.BuiltIn.BuiltIn</i> <i>method</i> ), 46
<code>get_child_handler()</code> ( <i>robot.result.xmllementhandlers.RootHandler</i> <i>method</i> ), 483	<code>get_current_date()</code> (in <i>module</i> <i>robot.libraries.DateTime</i> ), 75
<code>get_child_handler()</code> ( <i>robot.result.xmllementhandlers.StatisticsHandler</i> <i>method</i> ), 490	<code>get_dictionary_items()</code> ( <i>robot.libraries.Collections.Collections</i> <i>method</i> ), 68
<code>get_child_handler()</code> ( <i>robot.result.xmllementhandlers.StatusHandler</i> <i>method</i> ), 487	<code>get_dictionary_keys()</code> ( <i>robot.libraries.Collections.Collections</i> <i>method</i> ), 69
<code>get_child_handler()</code> ( <i>robot.result.xmllementhandlers.SuiteHandler</i> <i>method</i> ), 483	<code>get_dictionary_values()</code> ( <i>robot.libraries.Collections.Collections</i> <i>method</i> ), 69
<code>get_child_handler()</code> ( <i>robot.result.xmllementhandlers.TagHandler</i> <i>method</i> ), 488	<code>get_doc()</code> ( <i>robot.model.tagstatistics.TagStatInfo</i> <i>method</i> ), 247
<code>get_child_handler()</code> ( <i>robot.result.xmllementhandlers.TagsHandler</i> <i>method</i> ), 488	<code>get_element()</code> ( <i>robot.libraries.XML.XML</i> <i>method</i> ), 116
<code>get_child_handler()</code> ( <i>robot.result.xmllementhandlers.TestHandler</i> <i>method</i> ), 484	<code>get_element_attribute()</code> ( <i>robot.libraries.XML.XML</i> <i>method</i> ), 118
<code>get_child_handler()</code> ( <i>robot.result.xmllementhandlers.TimeoutHandler</i> <i>method</i> ), 489	<code>get_element_attributes()</code> ( <i>robot.libraries.XML.XML</i> <i>method</i> ), 118
<code>get_child_handler()</code> ( <i>robot.result.xmllementhandlers.TryHandler</i> <i>method</i> ), 485	<code>get_element_count()</code> ( <i>robot.libraries.XML.XML</i> <i>method</i> ), 117
	<code>get_element_text()</code> ( <i>robot.libraries.XML.XML</i> <i>method</i> ), 117
	<code>get_elements()</code> ( <i>robot.libraries.XML.XML</i> <i>method</i> ), 116
	<code>get_elements_texts()</code> ( <i>robot.libraries.XML.XML</i> <i>method</i> ), 117
	<code>get_environment_variable()</code> ( <i>robot.libraries.OperatingSystem.OperatingSystem</i> <i>method</i> ), 84
	<code>get_environment_variables()</code>



`(robot.libraries.OperatingSystem.OperatingSystem`  
`method), 85`  
`get_error_details()` (in module  
`robot.utils.error), 560`  
`get_error_message()` (in module  
`robot.utils.error), 560`  
`get_errors()` (`robot.errors.BreakLoop` method), 579  
`get_errors()` (`robot.errors.ContinueLoop` method),  
579  
`get_errors()` (`robot.errors.ExecutionFailed`  
`method), 576`  
`get_errors()` (`robot.errors.ExecutionFailures`  
`method), 577`  
`get_errors()` (`robot.errors.ExecutionPassed`  
`method), 578`  
`get_errors()` (`robot.errors.ExecutionStatus` method),  
576  
`get_errors()` (`robot.errors.HandlerExecutionFailed`  
`method), 577`  
`get_errors()` (`robot.errors.PassExecution` method),  
578  
`get_errors()` (`robot.errors.ReturnFromKeyword`  
`method), 579`  
`get_errors()` (`robot.errors.UserKeywordExecutionFailed`  
`method), 577`  
`get_file()` (`robot.libraries.OperatingSystem.OperatingSystem`  
`method), 79`  
`get_file_size()` (`robot.libraries.OperatingSystem.OperatingSystem`  
`method), 87`  
`get_from_dictionary()`  
`(robot.libraries.Collections.Collections`  
`method), 69`  
`get_from_list()` (`robot.libraries.Collections.Collections`  
`method), 69`  
`get_full_version()` (in module `robot.version`),  
586  
`get_host_info()` (`robot.libraries.Remote.TimeoutHTTPTransport`  
`method), 95`  
`get_host_info()` (`robot.libraries.Remote.TimeoutHTTPTransport`  
`method), 94`  
`get_index_from_list()`  
`(robot.libraries.Collections.Collections`  
`method), 69`  
`get_init_model()` (in module  
`robot.parsing.parser.parser), 356`  
`get_init_tokens()` (in module  
`robot.parsing.lexer.lexer), 290`  
`get_interpreter()` (in module `robot.version`), 586  
`get_keyword_arguments()`  
`(robot.libraries.Remote.Remote` method),  
94  
`get_keyword_arguments()`  
`(robot.libraries.Remote.XmlRpcRemoteClient`  
`method), 94`  
`get_keyword_documentation()`  
`(robot.libraries.Remote.Remote` method),  
94  
`get_keyword_documentation()`  
`(robot.libraries.Remote.XmlRpcRemoteClient`  
`method), 94`  
`get_keyword_names()`  
`(robot.libraries.Remote.Remote` method),  
94  
`get_keyword_names()`  
`(robot.libraries.Remote.XmlRpcRemoteClient`  
`method), 94`  
`get_keyword_names()`  
`(robot.libraries.Telnet.Telnet` method), 106  
`get_keyword_tags()`  
`(robot.libraries.Remote.Remote` method),  
94  
`get_keyword_tags()`  
`(robot.libraries.Remote.XmlRpcRemoteClient`  
`method), 94`  
`get_keyword_types()`  
`(robot.libraries.Remote.Remote` method),  
94  
`get_keyword_types()`  
`(robot.libraries.Remote.XmlRpcRemoteClient`  
`method), 94`  
`get_length()` (`robot.libraries.BuiltIn.BuiltIn`  
`method), 46`  
`get_library()` (`robot.running.namespace.KeywordStore`  
`method), 540`  
`get_library_information()`  
`(robot.libraries.Remote.XmlRpcRemoteClient`  
`method), 94`  
`get_library_instance()`  
`(robot.libraries.BuiltIn.BuiltIn` method),  
46  
`get_library_instance()`  
`(robot.running.namespace.Namespace`  
`method), 540`  
`get_library_instances()`  
`(robot.running.namespace.Namespace`  
`method), 540`  
`get_line()` (`robot.libraries.String.String` method), 99  
`get_line_count()` (`robot.libraries.String.String`  
`method), 99`  
`get_lines_containing_string()`  
`(robot.libraries.String.String` method), 99  
`get_lines_matching_pattern()`  
`(robot.libraries.String.String` method), 99  
`get_lines_matching_regexp()`  
`(robot.libraries.String.String` method), 99  
`get_link()` (`robot.model.tagstatistics.TagStatLink`  
`method), 247`  
`get_links()` (`robot.model.tagstatistics.TagStatInfo`

*method*), 247  
get\_match\_count() (*robot.libraries.Collections.Collections method*), 67  
get\_matches() (*robot.libraries.Collections.Collections method*), 67  
get\_message() (*robot.running.timeouts.KeywordTimeout method*), 510  
get\_message() (*robot.running.timeouts.TestTimeout method*), 510  
get\_model() (*in module robot.parsing.parser.parser*), 356  
get\_modified\_time() (*robot.libraries.OperatingSystem.OperatingSystem method*), 86  
get\_name() (*robot.output.pyloggingconf.RobotHandler method*), 277  
get\_process\_id() (*robot.libraries.Process.Process method*), 92  
get\_process\_object() (*robot.libraries.Process.Process method*), 92  
get\_process\_result() (*robot.libraries.Process.Process method*), 93  
get\_rebot\_settings() (*robot.conf.settings.RobotSettings method*), 31  
get\_regexp\_matches() (*robot.libraries.String.String method*), 100  
get\_resource\_model() (*in module robot.parsing.parser.parser*), 356  
get\_resource\_tokens() (*in module robot.parsing.lexer.lexer*), 290  
get\_runner() (*robot.running.namespace.KeywordStore method*), 540  
get\_runner() (*robot.running.namespace.Namespace method*), 540  
get\_selection\_from\_user() (*in module robot.libraries.Dialogs*), 77  
get\_selections\_from\_user() (*in module robot.libraries.Dialogs*), 77  
get\_shortcode\_from\_html() (*robot.libdocpkg.htmlutils.HtmlToText method*), 37  
get\_slice\_from\_list() (*robot.libraries.Collections.Collections method*), 69  
get\_socket() (*robot.libraries.Telnet.TelnetConnection method*), 110  
get\_stat() (*robot.model.tagstatistics.TagStatInfo method*), 247  
get\_substring() (*robot.libraries.String.String method*), 101  
get\_system\_encoding() (*in module robot.utils.encodingSniffer*), 560  
get\_time() (*robot.libraries.BuiltIn.BuiltIn method*), 47  
get\_token() (*robot.parsing.model.statements.Arguments method*), 334  
get\_token() (*robot.parsing.model.statements.Break method*), 350  
get\_token() (*robot.parsing.model.statements.Comment method*), 350  
get\_token() (*robot.parsing.model.statements.Continue method*), 349  
get\_token() (*robot.parsing.model.statements.DefaultTags method*), 321  
get\_token() (*robot.parsing.model.statements.Documentation method*), 319  
get\_token() (*robot.parsing.model.statements.DocumentationOrMetadata method*), 312  
get\_token() (*robot.parsing.model.statements.ElseHeader method*), 341  
get\_token() (*robot.parsing.model.statements.ElseIfHeader method*), 340  
get\_token() (*robot.parsing.model.statements.EmptyLine method*), 352  
get\_token() (*robot.parsing.model.statements.End method*), 345  
get\_token() (*robot.parsing.model.statements.Error method*), 351  
get\_token() (*robot.parsing.model.statements.ExceptHeader method*), 344  
get\_token() (*robot.parsing.model.statements.FinallyHeader method*), 345  
get\_token() (*robot.parsing.model.statements.Fixture method*), 315  
get\_token() (*robot.parsing.model.statements.ForceTags method*), 320  
get\_token() (*robot.parsing.model.statements.ForHeader method*), 337  
get\_token() (*robot.parsing.model.statements.IfElseHeader method*), 338  
get\_token() (*robot.parsing.model.statements.IfHeader method*), 339  
get\_token() (*robot.parsing.model.statements.InlineIfHeader method*), 340  
get\_token() (*robot.parsing.model.statements.KeywordCall method*), 335  
get\_token() (*robot.parsing.model.statements.KeywordName method*), 329  
get\_token() (*robot.parsing.model.statements.LibraryImport method*), 316  
get\_token() (*robot.parsing.model.statements.LoopControl method*), 348  
get\_token() (*robot.parsing.model.statements.Metadata method*), 320

`get_token()` (`robot.parsing.model.statements.MultiValueHeader` method), 314  
`get_token()` (`robot.parsing.model.statements.NoArgumentHeader` method), 342  
`get_token()` (`robot.parsing.model.statements.ResourceImport` method), 317  
`get_token()` (`robot.parsing.model.statements.ReturnStatement` method), 335  
`get_token()` (`robot.parsing.model.statements.ReturnStatement` method), 347  
`get_token()` (`robot.parsing.model.statements.SectionHeader` method), 315  
`get_token()` (`robot.parsing.model.statements.Setup` method), 330  
`get_token()` (`robot.parsing.model.statements.SingleValueHeader` method), 313  
`get_token()` (`robot.parsing.model.statements.Statement` method), 311  
`get_token()` (`robot.parsing.model.statements.SuiteSetup` method), 322  
`get_token()` (`robot.parsing.model.statements.SuiteTeardown` method), 323  
`get_token()` (`robot.parsing.model.statements.Tags` method), 331  
`get_token()` (`robot.parsing.model.statements.Teardown` method), 330  
`get_token()` (`robot.parsing.model.statements.TemplateArgument` method), 332  
`get_token()` (`robot.parsing.model.statements.TemplateArgument` method), 336  
`get_token()` (`robot.parsing.model.statements.TestCaseName` method), 328  
`get_token()` (`robot.parsing.model.statements.TestSetup` method), 324  
`get_token()` (`robot.parsing.model.statements.TestTeardown` method), 325  
`get_token()` (`robot.parsing.model.statements.TestTemplate` method), 326  
`get_token()` (`robot.parsing.model.statements.TestTimeout` method), 326  
`get_token()` (`robot.parsing.model.statements.Timeout` method), 333  
`get_token()` (`robot.parsing.model.statements.TryHeader` method), 343  
`get_token()` (`robot.parsing.model.statements.Variable` method), 327  
`get_token()` (`robot.parsing.model.statements.VariablesImport` method), 318  
`get_token()` (`robot.parsing.model.statements.WhileHeader` method), 346  
`get_tokens()` (`in module robot.parsing.lexer.lexer`), 290  
`get_tokens()` (`robot.parsing.model.statements.Arguments` method), 334  
`get_tokens()` (`robot.parsing.model.statements.Break` method), 350  
`get_tokens()` (`robot.parsing.model.statements.Comment` method), 350  
`get_tokens()` (`robot.parsing.model.statements.Continue` method), 349  
`get_tokens()` (`robot.parsing.model.statements.DefaultTags` method), 321  
`get_tokens()` (`robot.parsing.model.statements.Documentation` method), 319  
`get_tokens()` (`robot.parsing.model.statements.DocumentationOrMetadata` method), 312  
`get_tokens()` (`robot.parsing.model.statements.ElseHeader` method), 341  
`get_tokens()` (`robot.parsing.model.statements.ElseIfHeader` method), 341  
`get_tokens()` (`robot.parsing.model.statements.EmptyLine` method), 352  
`get_tokens()` (`robot.parsing.model.statements.End` method), 346  
`get_tokens()` (`robot.parsing.model.statements.Error` method), 351  
`get_tokens()` (`robot.parsing.model.statements.ExceptHeader` method), 344  
`get_tokens()` (`robot.parsing.model.statements.FinallyHeader` method), 345  
`get_tokens()` (`robot.parsing.model.statements.Fixture` method), 315  
`get_tokens()` (`robot.parsing.model.statements.ForceTags` method), 321  
`get_tokens()` (`robot.parsing.model.statements.ForHeader` method), 337  
`get_tokens()` (`robot.parsing.model.statements.IfElseHeader` method), 338  
`get_tokens()` (`robot.parsing.model.statements.IfHeader` method), 339  
`get_tokens()` (`robot.parsing.model.statements.InlineIfHeader` method), 340  
`get_tokens()` (`robot.parsing.model.statements.KeywordCall` method), 336  
`get_tokens()` (`robot.parsing.model.statements.KeywordName` method), 329  
`get_tokens()` (`robot.parsing.model.statements.LibraryImport` method), 316  
`get_tokens()` (`robot.parsing.model.statements.LoopControl` method), 348  
`get_tokens()` (`robot.parsing.model.statements.Metadata` method), 320  
`get_tokens()` (`robot.parsing.model.statements.MultiValueHeader` method), 314  
`get_tokens()` (`robot.parsing.model.statements.NoArgumentHeader` method), 342

`get_tokens()` (`robot.parsing.model.statements.ResourceImport` method), 317

`get_tokens()` (`robot.parsing.model.statements.ReturnStatement` method), 335

`get_tokens()` (`robot.parsing.model.statements.ReturnStatementOrMetadata` method), 347

`get_tokens()` (`robot.parsing.model.statements.SectionHeader` method), 316

`get_tokens()` (`robot.parsing.model.statements.Setup` method), 330

`get_tokens()` (`robot.parsing.model.statements.SingleValue` method), 313

`get_tokens()` (`robot.parsing.model.statements.Statement` method), 311

`get_tokens()` (`robot.parsing.model.statements.SuiteSetup` method), 322

`get_tokens()` (`robot.parsing.model.statements.SuiteTeardown` method), 323

`get_tokens()` (`robot.parsing.model.statements.Tags` method), 331

`get_tokens()` (`robot.parsing.model.statements.Teardown` method), 331

`get_tokens()` (`robot.parsing.model.statements.TemplateArgument` method), 332

`get_tokens()` (`robot.parsing.model.statements.TemplateArgumentList` method), 336

`get_tokens()` (`robot.parsing.model.statements.TestCaseName` method), 328

`get_tokens()` (`robot.parsing.model.statements.TestSetup` method), 324

`get_tokens()` (`robot.parsing.model.statements.TestTeardown` method), 325

`get_tokens()` (`robot.parsing.model.statements.TestTemplate` method), 326

`get_tokens()` (`robot.parsing.model.statements.TestTimeout` method), 326

`get_tokens()` (`robot.parsing.model.statements.Timeout` method), 333

`get_tokens()` (`robot.parsing.model.statements.TryHeader` method), 343

`get_tokens()` (`robot.parsing.model.statements.VariableSetup` method), 327

`get_tokens()` (`robot.parsing.model.statements.VariableStatement` method), 318

`get_tokens()` (`robot.parsing.model.statements.WhileHeader` method), 346

`get_value()` (`robot.parsing.model.statements.Arguments` method), 334

`get_value()` (`robot.parsing.model.statements.Break` method), 350

`get_value()` (`robot.parsing.model.statements.Comment` method), 351

`get_value()` (`robot.parsing.model.statements.Continue` method), 349

`get_value()` (`robot.parsing.model.statements.DefaultTags` method), 321

`get_value()` (`robot.parsing.model.statements.Documentation` method), 319

`get_value()` (`robot.parsing.model.statements.DocumentationOrMetadata` method), 312

`get_value()` (`robot.parsing.model.statements.ElseHeader` method), 341

`get_value()` (`robot.parsing.model.statements.ElseIfHeader` method), 341

`get_value()` (`robot.parsing.model.statements.EmptyLine` method), 352

`get_value()` (`robot.parsing.model.statements.End` method), 346

`get_value()` (`robot.parsing.model.statements.Error` method), 351

`get_value()` (`robot.parsing.model.statements.ExceptHeader` method), 344

`get_value()` (`robot.parsing.model.statements.FinallyHeader` method), 345

`get_value()` (`robot.parsing.model.statements.Fixture` method), 315

`get_value()` (`robot.parsing.model.statements.ForceTags` method), 321

`get_value()` (`robot.parsing.model.statements.ForHeader` method), 337

`get_value()` (`robot.parsing.model.statements.IfElseHeader` method), 338

`get_value()` (`robot.parsing.model.statements.IfHeader` method), 339

`get_value()` (`robot.parsing.model.statements.InlineIfHeader` method), 340

`get_value()` (`robot.parsing.model.statements.KeywordCall` method), 336

`get_value()` (`robot.parsing.model.statements.KeywordName` method), 329

`get_value()` (`robot.parsing.model.statements.LibraryImport` method), 316

`get_value()` (`robot.parsing.model.statements.LoopControl` method), 348

`get_value()` (`robot.parsing.model.statements.Metadata` method), 320

`get_value()` (`robot.parsing.model.statements.MultiValue` method), 314

`get_value()` (`robot.parsing.model.statements.NoArgumentHeader` method), 342

`get_value()` (`robot.parsing.model.statements.ResourceImport` method), 317

`get_value()` (`robot.parsing.model.statements.Return` method), 335

`get_value()` (`robot.parsing.model.statements.ReturnStatement` method), 347

`get_value()` (`robot.parsing.model.statements.SectionHeader` method), 316



`get_value()` (`robot.parsing.model.statements.Setup` method), 330  
`get_value()` (`robot.parsing.model.statements.SingleValue` method), 313  
`get_value()` (`robot.parsing.model.statements.Statement` method), 312  
`get_value()` (`robot.parsing.model.statements.SuiteSetup` method), 322  
`get_value()` (`robot.parsing.model.statements.SuiteTeardown` method), 323  
`get_value()` (`robot.parsing.model.statements.Tags` method), 331  
`get_value()` (`robot.parsing.model.statements.Teardown` method), 331  
`get_value()` (`robot.parsing.model.statements.Template` method), 332  
`get_value()` (`robot.parsing.model.statements.TemplateArgument` method), 336  
`get_value()` (`robot.parsing.model.statements.TestCaseName` method), 328  
`get_value()` (`robot.parsing.model.statements.TestSetup` method), 324  
`get_value()` (`robot.parsing.model.statements.TestTeardown` method), 325  
`get_value()` (`robot.parsing.model.statements.TestTemplate` method), 326  
`get_value()` (`robot.parsing.model.statements.TestTimeout` method), 326  
`get_value()` (`robot.parsing.model.statements.Timeout` method), 333  
`get_value()` (`robot.parsing.model.statements.TryHeader` method), 343  
`get_value()` (`robot.parsing.model.statements.Variable` method), 327  
`get_value()` (`robot.parsing.model.statements.VariablesImport` method), 318  
`get_value()` (`robot.parsing.model.statements.WhileHeader` method), 346  
`get_value_from_user()` (in `robot.libraries.Dialogs` module), 77  
`get_values()` (`robot.parsing.model.statements.Arguments` method), 334  
`get_values()` (`robot.parsing.model.statements.Break` method), 350  
`get_values()` (`robot.parsing.model.statements.Comment` method), 351  
`get_values()` (`robot.parsing.model.statements.Continue` method), 349  
`get_values()` (`robot.parsing.model.statements.DefaultTags` method), 321  
`get_values()` (`robot.parsing.model.statements.Documentation` method), 319  
`get_values()` (`robot.parsing.model.statements.DocumentationMetadata` method), 312  
`get_values()` (`robot.parsing.model.statements.ElseHeader` method), 342  
`get_values()` (`robot.parsing.model.statements.ElseIfHeader` method), 341  
`get_values()` (`robot.parsing.model.statements.EmptyLine` method), 352  
`get_values()` (`robot.parsing.model.statements.End` method), 346  
`get_values()` (`robot.parsing.model.statements.Error` method), 352  
`get_values()` (`robot.parsing.model.statements.ExceptHeader` method), 344  
`get_values()` (`robot.parsing.model.statements.FinallyHeader` method), 345  
`get_values()` (`robot.parsing.model.statements.Fixture` method), 315  
`get_values()` (`robot.parsing.model.statements.ForceTags` method), 321  
`get_values()` (`robot.parsing.model.statements.ForHeader` method), 337  
`get_values()` (`robot.parsing.model.statements.IfElseHeader` method), 338  
`get_values()` (`robot.parsing.model.statements.IfHeader` method), 339  
`get_values()` (`robot.parsing.model.statements.InlineIfHeader` method), 340  
`get_values()` (`robot.parsing.model.statements.KeywordCall` method), 336  
`get_values()` (`robot.parsing.model.statements.KeywordName` method), 329  
`get_values()` (`robot.parsing.model.statements.LibraryImport` method), 317  
`get_values()` (`robot.parsing.model.statements.LoopControl` method), 348  
`get_values()` (`robot.parsing.model.statements.Metadata` method), 320  
`get_values()` (`robot.parsing.model.statements.MultiValue` method), 314  
`get_values()` (`robot.parsing.model.statements.NoArgumentHeader` method), 342  
`get_values()` (`robot.parsing.model.statements.ResourceImport` method), 317  
`get_values()` (`robot.parsing.model.statements.Return` method), 335  
`get_values()` (`robot.parsing.model.statements.ReturnStatement` method), 347  
`get_values()` (`robot.parsing.model.statements.SectionHeader` method), 316  
`get_values()` (`robot.parsing.model.statements.Setup` method), 330  
`get_values()` (`robot.parsing.model.statements.SingleValue` method), 313  
`get_values()` (`robot.parsing.model.statements.Statement` method), 312

`get_values()` (`robot.parsing.model.statements.SuiteSetup` method), 322  
`get_values()` (`robot.parsing.model.statements.SuiteTeardown` method), 323  
`get_values()` (`robot.parsing.model.statements.Tags` method), 331  
`get_values()` (`robot.parsing.model.statements.Teardown` method), 331  
`get_values()` (`robot.parsing.model.statements.Template` method), 332  
`get_values()` (`robot.parsing.model.statements.TemplateArguments` method), 336  
`get_values()` (`robot.parsing.model.statements.TestCaseName` method), 328  
`get_values()` (`robot.parsing.model.statements.TestSetup` method), 324  
`get_values()` (`robot.parsing.model.statements.TestTeardown` method), 325  
`get_values()` (`robot.parsing.model.statements.TestTemplate` method), 326  
`get_values()` (`robot.parsing.model.statements.TestTimeout` method), 327  
`get_values()` (`robot.parsing.model.statements.Timeout` method), 333  
`get_values()` (`robot.parsing.model.statements.TryHeader` method), 343  
`get_values()` (`robot.parsing.model.statements.Variable` method), 327  
`get_values()` (`robot.parsing.model.statements.VariableSupport` method), 318  
`get_values()` (`robot.parsing.model.statements.WhileHeader` method), 347  
`get_variable_value()` (`robot.libraries.BuiltIn.BuiltIn` method), 47  
`get_variables()` (`robot.libraries.BuiltIn.BuiltIn` method), 48  
`get_version()` (in module `robot.version`), 586  
`getboolean()` (`robot.libraries.dialogs_py.InputDialog` method), 140  
`getboolean()` (`robot.libraries.dialogs_py.MessageDialog` method), 126  
`getboolean()` (`robot.libraries.dialogs_py.MultipleSelectionDialog` method), 168  
`getboolean()` (`robot.libraries.dialogs_py.PassFailDialog` method), 182  
`getboolean()` (`robot.libraries.dialogs_py.SelectionDialog` method), 154  
`getdouble()` (`robot.libraries.dialogs_py.InputDialog` method), 140  
`getdouble()` (`robot.libraries.dialogs_py.MessageDialog` method), 126  
`getdouble()` (`robot.libraries.dialogs_py.MultipleSelectionDialog` method), 168  
`getdouble()` (`robot.libraries.dialogs_py.PassFailDialog` method), 182  
`getdouble()` (`robot.libraries.dialogs_py.SelectionDialog` method), 154  
`getint()` (`robot.libraries.dialogs_py.InputDialog` method), 140  
`getint()` (`robot.libraries.dialogs_py.MessageDialog` method), 126  
`getint()` (`robot.libraries.dialogs_py.MultipleSelectionDialog` method), 168  
`getint()` (`robot.libraries.dialogs_py.PassFailDialog` method), 182  
`getint()` (`robot.libraries.dialogs_py.SelectionDialog` method), 154  
`getkeywordarguments` (class in `robot.running.dynamicmethods`), 513  
`getkeyworddocumentation` (class in `robot.running.dynamicmethods`), 513  
`getkeywordnames` (class in `robot.running.dynamicmethods`), 513  
`getkeywordsource` (class in `robot.running.dynamicmethods`), 513  
`getkeywordtags` (class in `robot.running.dynamicmethods`), 513  
`getkeywordtypes` (class in `robot.running.dynamicmethods`), 513  
`getparser()` (`robot.libraries.Remote.TimeoutHTTPSTransport` method), 95  
`getparser()` (`robot.libraries.Remote.TimeoutHTTPSTransport` method), 95  
`getvar()` (`robot.libraries.dialogs_py.InputDialog` method), 140  
`getvar()` (`robot.libraries.dialogs_py.MessageDialog` method), 126  
`getvar()` (`robot.libraries.dialogs_py.MultipleSelectionDialog` method), 168  
`getvar()` (`robot.libraries.dialogs_py.PassFailDialog` method), 182  
`getvar()` (`robot.libraries.dialogs_py.SelectionDialog` method), 154  
`glob_escape()` (in module `robot.utils.escaping`), 560  
`GlobalScope` (class in `robot.running.libraryscopes`), 515  
`GlobalVariables` (class in `robot.variables.scopes`), 571  
`GlobalVariableValue` (class in `robot.variables.resolvable`), 570  
`grab_current()` (`robot.libraries.dialogs_py.InputDialog` method), 140  
`grab_current()` (`robot.libraries.dialogs_py.MessageDialog` method), 126  
`grab_current()` (`robot.libraries.dialogs_py.MultipleSelectionDialog` method), 168  
`grab_current()` (`robot.libraries.dialogs_py.PassFailDialog` method), 182

651

[method](#)), 183

[grid\\_location\(\)](#) ([robot.libraries.dialogs\\_py.SelectionDialog](#) [method](#)), 155

[grid\\_propagate\(\)](#) ([robot.libraries.dialogs\\_py.InputDialog](#) [method](#)), 141

[grid\\_propagate\(\)](#) ([robot.libraries.dialogs\\_py.MessageDialog](#) [method](#)), 127

[grid\\_propagate\(\)](#) ([robot.libraries.dialogs\\_py.MultipleSelectionDialog](#) [method](#)), 169

[grid\\_propagate\(\)](#) ([robot.libraries.dialogs\\_py.PassFailDialog](#) [method](#)), 183

[grid\\_propagate\(\)](#) ([robot.libraries.dialogs\\_py.SelectionDialog](#) [method](#)), 155

[grid\\_rowconfigure\(\)](#) ([robot.libraries.dialogs\\_py.InputDialog](#) [method](#)), 141

[grid\\_rowconfigure\(\)](#) ([robot.libraries.dialogs\\_py.MessageDialog](#) [method](#)), 127

[grid\\_rowconfigure\(\)](#) ([robot.libraries.dialogs\\_py.MultipleSelectionDialog](#) [method](#)), 169

[grid\\_rowconfigure\(\)](#) ([robot.libraries.dialogs\\_py.PassFailDialog](#) [method](#)), 183

[grid\\_rowconfigure\(\)](#) ([robot.libraries.dialogs\\_py.SelectionDialog](#) [method](#)), 155

[grid\\_size\(\)](#) ([robot.libraries.dialogs\\_py.InputDialog](#) [method](#)), 141

[grid\\_size\(\)](#) ([robot.libraries.dialogs\\_py.MessageDialog](#) [method](#)), 127

[grid\\_size\(\)](#) ([robot.libraries.dialogs\\_py.MultipleSelectionDialog](#) [method](#)), 169

[grid\\_size\(\)](#) ([robot.libraries.dialogs\\_py.PassFailDialog](#) [method](#)), 183

[grid\\_size\(\)](#) ([robot.libraries.dialogs\\_py.SelectionDialog](#) [method](#)), 155

[grid\\_slaves\(\)](#) ([robot.libraries.dialogs\\_py.InputDialog](#) [method](#)), 141

[grid\\_slaves\(\)](#) ([robot.libraries.dialogs\\_py.MessageDialog](#) [method](#)), 127

[grid\\_slaves\(\)](#) ([robot.libraries.dialogs\\_py.MultipleSelectionDialog](#) [method](#)), 169

[grid\\_slaves\(\)](#) ([robot.libraries.dialogs\\_py.PassFailDialog](#) [method](#)), 183

[grid\\_slaves\(\)](#) ([robot.libraries.dialogs\\_py.SelectionDialog](#) [method](#)), 155

[group\(\)](#) ([robot.libraries.dialogs\\_py.InputDialog](#) [method](#)), 141

[group\(\)](#) ([robot.libraries.dialogs\\_py.MessageDialog](#) [method](#)), 127

[group\(\)](#) ([robot.libraries.dialogs\\_py.MultipleSelectionDialog](#) [method](#)), 169

[group\(\)](#) ([robot.libraries.dialogs\\_py.PassFailDialog](#) [method](#)), 183

[group\(\)](#) ([robot.libraries.dialogs\\_py.SelectionDialog](#) [method](#)), 155

[handle\(\)](#) ([robot.output.pyloggingconf.RobotHandler](#) [method](#)), 277

[handle\(\)](#) ([robot.running.arguments.argumentresolver.DictToKwargs](#) [method](#)), 493

[handle\\_imports\(\)](#) ([robot.running.namespace.Namespace](#) [method](#)), 540

[handle\\_suite\\_teardown\\_failures\(\)](#) ([robot.result.executionresult.CombinedResult](#) [method](#)), 378

[handle\\_suite\\_teardown\\_failures\(\)](#) ([robot.result.executionresult.Result](#) [method](#)), 378

[handle\\_suite\\_teardown\\_failures\(\)](#) ([robot.result.model.TestSuite](#) [method](#)), 463

[handle\\_error\(\)](#) ([robot.output.pyloggingconf.RobotHandler](#) [method](#)), 277

[Handler\(\)](#) (in module [robot.running.handlers](#)), 513

[HandlerExecutionFailed](#), 577

[HandlerStore](#) (class in [robot.running.handlerstore](#)), 514

[handles\(\)](#) ([robot.htmldata.htmlfilewriter.CssFileWriter](#) [method](#)), 34

[handles\(\)](#) ([robot.htmldata.htmlfilewriter.GeneratorWriter](#) [method](#)), 34

[handles\(\)](#) ([robot.htmldata.htmlfilewriter JsFileWriter](#) [method](#)), 34

[handles\(\)](#) ([robot.htmldata.htmlfilewriter.LineWriter](#) [method](#)), 34

[handles\(\)](#) ([robot.htmldata.htmlfilewriter.ModelWriter](#) [method](#)), 34

[handles\(\)](#) ([robot.htmldata.jsonwriter.DictDumper](#) [method](#)), 35

[handles\(\)](#) ([robot.htmldata.jsonwriter.IntegerDumper](#) [method](#)), 34

[handles\(\)](#) ([robot.htmldata.jsonwriter.MappingDumper](#) [method](#)), 35

[handles\(\)](#) ([robot.htmldata.jsonwriter.NoneDumper](#) [method](#)), 35

[handles\(\)](#) ([robot.htmldata.jsonwriter.StringDumper](#) [method](#)), 34

[handles\(\)](#) ([robot.htmldata.jsonwriter.TupleListDumper](#) [method](#)), 35

[handles\(\)](#) ([robot.libdocpkg.consoleviewer.ConsoleViewer](#) [class method](#)), 36

[handles\(\)](#) ([robot.libdocpkg.htmlwriter.LibdocModelWriter](#) [method](#)), 37

[handles\(\)](#) ([robot.parsing.lexer.blocklexers.BlockLexer](#) [method](#)), 283



[handles\(\) \(robot.parsing.lexer.blocklexers.CommentSectionLexer method\), 285](#)  
[handles\(\) \(robot.parsing.lexer.blocklexers.ErrorSectionLexer method\), 285](#)  
[handles\(\) \(robot.parsing.lexer.blocklexers.FileLexer method\), 284](#)  
[handles\(\) \(robot.parsing.lexer.blocklexers.ForLexer method\), 287](#)  
[handles\(\) \(robot.parsing.lexer.blocklexers.IfLexer method\), 287](#)  
[handles\(\) \(robot.parsing.lexer.blocklexers.ImplicitCommentSectionLexer method\), 285](#)  
[handles\(\) \(robot.parsing.lexer.blocklexers.InlineIfLexer method\), 287](#)  
[handles\(\) \(robot.parsing.lexer.blocklexers.KeywordLexer method\), 286](#)  
[handles\(\) \(robot.parsing.lexer.blocklexers.KeywordSectionLexer method\), 285](#)  
[handles\(\) \(robot.parsing.lexer.blocklexers.NestedBlockLexer method\), 287](#)  
[handles\(\) \(robot.parsing.lexer.blocklexers.SectionLexer method\), 284](#)  
[handles\(\) \(robot.parsing.lexer.blocklexers.SettingSectionLexer method\), 284](#)  
[handles\(\) \(robot.parsing.lexer.blocklexers.TestCaseLexer method\), 286](#)  
[handles\(\) \(robot.parsing.lexer.blocklexers.TestCaseSectionLexer method\), 285](#)  
[handles\(\) \(robot.parsing.lexer.blocklexers.TestOrKeywordLexer method\), 286](#)  
[handles\(\) \(robot.parsing.lexer.blocklexers.TryLexer method\), 288](#)  
[handles\(\) \(robot.parsing.lexer.blocklexers.VariableSectionLexer method\), 284](#)  
[handles\(\) \(robot.parsing.lexer.blocklexers.WhileLexer method\), 287](#)  
[handles\(\) \(robot.parsing.lexer.statementlexers.BreakLexer method\), 299](#)  
[handles\(\) \(robot.parsing.lexer.statementlexers.CommentLexer method\), 295](#)  
[handles\(\) \(robot.parsing.lexer.statementlexers.CommentSectionHeaderLexer method\), 295](#)  
[handles\(\) \(robot.parsing.lexer.statementlexers.ContinueLexer method\), 298](#)  
[handles\(\) \(robot.parsing.lexer.statementlexers.ElseHeaderLexer method\), 297](#)  
[handles\(\) \(robot.parsing.lexer.statementlexers.ElselfHeaderLexer method\), 297](#)  
[handles\(\) \(robot.parsing.lexer.statementlexers.EndLexer method\), 298](#)  
[handles\(\) \(robot.parsing.lexer.statementlexers.ErrorSectionHeaderLexer method\), 295](#)  
[handles\(\) \(robot.parsing.lexer.statementlexers.ExceptHeaderLexer method\), 297](#)  
[handles\(\) \(robot.parsing.lexer.statementlexers.FinallyHeaderLexer method\), 298](#)  
[handles\(\) \(robot.parsing.lexer.statementlexers.ForHeaderLexer method\), 296](#)  
[handles\(\) \(robot.parsing.lexer.statementlexers.IfHeaderLexer method\), 297](#)  
[handles\(\) \(robot.parsing.lexer.statementlexers.InlineIfHeaderLexer method\), 297](#)  
[handles\(\) \(robot.parsing.lexer.statementlexers.KeywordCallLexer method\), 296](#)  
[handles\(\) \(robot.parsing.lexer.statementlexers.KeywordSectionHeaderLexer method\), 295](#)  
[handles\(\) \(robot.parsing.lexer.statementlexers.Lexer method\), 293](#)  
[handles\(\) \(robot.parsing.lexer.statementlexers.ReturnLexer method\), 298](#)  
[handles\(\) \(robot.parsing.lexer.statementlexers.SectionHeaderLexer method\), 294](#)  
[handles\(\) \(robot.parsing.lexer.statementlexers.SettingLexer method\), 296](#)  
[handles\(\) \(robot.parsing.lexer.statementlexers.SettingSectionHeaderLexer method\), 294](#)  
[handles\(\) \(robot.parsing.lexer.statementlexers.SingleType method\), 294](#)  
[handles\(\) \(robot.parsing.lexer.statementlexers.StatementLexer method\), 294](#)  
[handles\(\) \(robot.parsing.lexer.statementlexers.TestCaseSectionHeaderLexer method\), 295](#)  
[handles\(\) \(robot.parsing.lexer.statementlexers.TestOrKeywordSettingLexer method\), 296](#)  
[handles\(\) \(robot.parsing.lexer.statementlexers.TryHeaderLexer method\), 297](#)  
[handles\(\) \(robot.parsing.lexer.statementlexers.TypeAndArguments method\), 294](#)  
[handles\(\) \(robot.parsing.lexer.statementlexers.VariableLexer method\), 296](#)  
[handles\(\) \(robot.parsing.lexer.statementlexers.VariableSectionHeaderLexer method\), 295](#)  
[handles\(\) \(robot.parsing.lexer.statementlexers.WhileHeaderLexer method\), 298](#)  
[handles\(\) \(robot.parsing.parser.blockparsers.BlockParser method\), 353](#)  
[handles\(\) \(robot.parsing.parser.blockparsers.ForParser method\), 354](#)  
[handles\(\) \(robot.parsing.parser.blockparsers.IfParser method\), 354](#)  
[handles\(\) \(robot.parsing.parser.blockparsers.KeywordParser method\), 354](#)  
[handles\(\) \(robot.parsing.parser.blockparsers.NestedBlockParser method\), 354](#)  
[handles\(\) \(robot.parsing.parser.blockparsers.Parser method\), 353](#)  
[handles\(\) \(robot.parsing.parser.blockparsers.TestCaseParser method\), 353](#)

`handles()` (`robot.parsing.parser.blockparsers.TryParser` method), 354

`handles()` (`robot.parsing.parser.blockparsers.WhileParser` method), 354

`handles()` (`robot.parsing.parser.fileparser.CommentSectionParser` method), 355

`handles()` (`robot.parsing.parser.fileparser.FileParser` method), 354

`handles()` (`robot.parsing.parser.fileparser.ImplicitCommentSectionParser` method), 355

`handles()` (`robot.parsing.parser.fileparser.KeywordSectionParser` method), 355

`handles()` (`robot.parsing.parser.fileparser.SectionParser` method), 355

`handles()` (`robot.parsing.parser.fileparser.SettingSectionParser` method), 355

`handles()` (`robot.parsing.parser.fileparser.TestCaseSectionParser` method), 355

`handles()` (`robot.parsing.parser.fileparser.VariableSectionParser` method), 355

`handles()` (`robot.reporting.logreportwriters.RobotModelWriter` method), 359

`handles()` (`robot.running.arguments.typeconverters.BooleanConverter` class method), 497

`handles()` (`robot.running.arguments.typeconverters.ByteArrayConverter` class method), 499

`handles()` (`robot.running.arguments.typeconverters.BytesConverter` class method), 498

`handles()` (`robot.running.arguments.typeconverters.CombinedConverter` class method), 502

`handles()` (`robot.running.arguments.typeconverters.CustomConverter` class method), 503

`handles()` (`robot.running.arguments.typeconverters.DateConverter` class method), 499

`handles()` (`robot.running.arguments.typeconverters.DateTimeConverter` class method), 499

`handles()` (`robot.running.arguments.typeconverters.DecimalConverter` class method), 498

`handles()` (`robot.running.arguments.typeconverters.DictionaryConverter` class method), 501

`handles()` (`robot.running.arguments.typeconverters.EnumConverter` class method), 496

`handles()` (`robot.running.arguments.typeconverters.FloodGateConverter` class method), 497

`handles()` (`robot.running.arguments.typeconverters.FrozenSetConverter` class method), 502

`handles()` (`robot.running.arguments.typeconverters.IntegerConverter` class method), 497

`handles()` (`robot.running.arguments.typeconverters.ListConverter` class method), 501

`handles()` (`robot.running.arguments.typeconverters.NonEmptyConverter` class method), 500

`handles()` (`robot.running.arguments.typeconverters.SetConverter` class method), 502

`handles()` (`robot.running.arguments.typeconverters.StringConverter` class method), 496

`handles()` (`robot.running.arguments.typeconverters.TimeDeltaConverter` class method), 500

`handles()` (`robot.running.arguments.typeconverters.TupleConverter` class method), 501

`handles()` (`robot.running.arguments.typeconverters.TypeConverter` class method), 495

`handles()` (`robot.testdoc.TestdocModelWriter` method), 585

`handles()` (`robot.utils.htmlformatters.HeaderFormatter` method), 562

`handles()` (`robot.utils.htmlformatters.LineFormatter` method), 561

`handles()` (`robot.utils.htmlformatters.ListFormatter` method), 562

`handles()` (`robot.utils.htmlformatters.ParagraphFormatter` method), 562

`handles()` (`robot.utils.htmlformatters.PreformattedFormatter` method), 562

`handles()` (`robot.utils.htmlformatters.RulerFormatter` method), 561

`handles()` (`robot.utils.htmlformatters.TableFormatter` method), 562

`handles()` (`robot.utils.importer.ByPathImporter` method), 563

`handles()` (`robot.utils.importer.DottedImporter` method), 564

`handles()` (`robot.utils.importer.NonDottedImporter` method), 563

`handles()` (`robot.parsing.model.statements.Arguments` attribute), 334

`handles()` (`robot.parsing.model.statements.Break` attribute), 350

`handles()` (`robot.parsing.model.statements.Comment` attribute), 351

`handles()` (`robot.parsing.model.statements.Continue` attribute), 349

`handles()` (`robot.parsing.model.statements.DefaultTags` attribute), 322

`handles()` (`robot.parsing.model.statements.Documentation` attribute), 319

`handles()` (`robot.parsing.model.statements.DocumentationOrMeta` attribute), 312

`handles()` (`robot.parsing.model.statements.ElseHeader` attribute), 342

`handles()` (`robot.parsing.model.statements.ElseIfHeader` attribute), 341

`handles()` (`robot.parsing.model.statements.EmptyLine` attribute), 352

`handles()` (`robot.parsing.model.statements.End` attribute), 346

`handles()` (`robot.parsing.model.statements.Error` attribute), 351

[handles\\_types \(robot.parsing.model.statements.ExceptHeader attribute\), 344](#)  
[handles\\_types \(robot.parsing.model.statements.FinallyHeader attribute\), 345](#)  
[handles\\_types \(robot.parsing.model.statements.FixtureHeader attribute\), 315](#)  
[handles\\_types \(robot.parsing.model.statements.ForceTags attribute\), 321](#)  
[handles\\_types \(robot.parsing.model.statements.ForHeader attribute\), 337](#)  
[handles\\_types \(robot.parsing.model.statements.IfElseHeader attribute\), 338](#)  
[handles\\_types \(robot.parsing.model.statements.IfHeader attribute\), 339](#)  
[handles\\_types \(robot.parsing.model.statements.InlineIfHeader attribute\), 340](#)  
[handles\\_types \(robot.parsing.model.statements.KeywordCall attribute\), 336](#)  
[handles\\_types \(robot.parsing.model.statements.KeywordName attribute\), 329](#)  
[handles\\_types \(robot.parsing.model.statements.LibraryImport attribute\), 317](#)  
[handles\\_types \(robot.parsing.model.statements.LoopControl attribute\), 348](#)  
[handles\\_types \(robot.parsing.model.statements.Metadata attribute\), 320](#)  
[handles\\_types \(robot.parsing.model.statements.MultiValue attribute\), 314](#)  
[handles\\_types \(robot.parsing.model.statements.NoArgumentHeader attribute\), 342](#)  
[handles\\_types \(robot.parsing.model.statements.ResourceImport attribute\), 317](#)  
[handles\\_types \(robot.parsing.model.statements.Return attribute\), 335](#)  
[handles\\_types \(robot.parsing.model.statements.ReturnStatement attribute\), 347](#)  
[handles\\_types \(robot.parsing.model.statements.SectionHeader attribute\), 315](#)  
[handles\\_types \(robot.parsing.model.statements.Setup attribute\), 330](#)  
[handles\\_types \(robot.parsing.model.statements.SingleValue attribute\), 313](#)  
[handles\\_types \(robot.parsing.model.statements.Statement attribute\), 311](#)  
[handles\\_types \(robot.parsing.model.statements.SuiteSetup attribute\), 322](#)  
[handles\\_types \(robot.parsing.model.statements.SuiteTeardown attribute\), 323](#)  
[handles\\_types \(robot.parsing.model.statements.Tags attribute\), 332](#)  
[handles\\_types \(robot.parsing.model.statements.Teardown attribute\), 331](#)  
[handles\\_types \(robot.parsing.model.statements.Template attribute\), 332](#)  
[handles\\_types \(robot.parsing.model.statements.TemplateArguments attribute\), 336](#)  
[handles\\_types \(robot.parsing.model.statements.TestCaseName attribute\), 328](#)  
[handles\\_types \(robot.parsing.model.statements.TestSetup attribute\), 324](#)  
[handles\\_types \(robot.parsing.model.statements.TestTeardown attribute\), 325](#)  
[handles\\_types \(robot.parsing.model.statements.TestTemplate attribute\), 326](#)  
[handles\\_types \(robot.parsing.model.statements.TestTimeout attribute\), 327](#)  
[handles\\_types \(robot.parsing.model.statements.Timeout attribute\), 333](#)  
[handles\\_types \(robot.parsing.model.statements.TryHeader attribute\), 343](#)  
[handles\\_types \(robot.parsing.model.statements.Variable attribute\), 327](#)  
[handles\\_types \(robot.parsing.model.statements.VariablesImport attribute\), 318](#)  
[handles\\_types \(robot.parsing.model.statements.WhileHeader attribute\), 347](#)  
[has\\_setup \(robot.model.body.BodyItem attribute\), 193](#)  
[has\\_setup \(robot.model.control.Break attribute\), 215](#)  
[has\\_setup \(robot.model.control.Continue attribute\), 214](#)  
[has\\_setup \(robot.model.control.For attribute\), 205](#)  
[has\\_setup \(robot.model.control.If attribute\), 208](#)  
[has\\_setup \(robot.model.control.IfBranch attribute\), 207](#)  
[has\\_setup \(robot.model.control.Return attribute\), 212](#)  
[has\\_setup \(robot.model.control.Try attribute\), 211](#)  
[has\\_setup \(robot.model.control.TryBranch attribute\), 210](#)  
[has\\_setup \(robot.model.control.While attribute\), 206](#)  
[has\\_setup \(robot.model.keyword.Keyword attribute\), 226](#)  
[has\\_setup \(robot.model.message.Message attribute\), 229](#)  
[has\\_setup \(robot.model.testcase.TestCase attribute\), 248](#)  
[has\\_setup \(robot.model.testsuite.TestSuite attribute\), 251](#)  
[has\\_setup \(robot.output.loggerhelper.Message attribute\), 275](#)  
[has\\_setup \(robot.result.model.Break attribute\), 453](#)  
[has\\_setup \(robot.result.model.Continue attribute\), 451](#)  
[has\\_setup \(robot.result.model.For attribute\), 432](#)  
[has\\_setup \(robot.result.model.ForIteration attribute\), 430](#)  
[has\\_setup \(robot.result.model.If attribute\), 441](#)  
[has\\_setup \(robot.result.model.IfBranch attribute\), 439](#)  
[has\\_setup \(robot.result.model.Keyword attribute\), 456](#)

`has_setup` (*robot.result.model.Message* attribute), 428  
`has_setup` (*robot.result.model.Return* attribute), 448  
`has_setup` (*robot.result.model.TestCase* attribute), 458  
`has_setup` (*robot.result.model.TestSuite* attribute), 461  
`has_setup` (*robot.result.model.Try* attribute), 446  
`has_setup` (*robot.result.model.TryBranch* attribute), 444  
`has_setup` (*robot.result.model.While* attribute), 437  
`has_setup` (*robot.result.model.WhileIteration* attribute), 435  
`has_setup` (*robot.running.model.Break* attribute), 532  
`has_setup` (*robot.running.model.Continue* attribute), 531  
`has_setup` (*robot.running.model.For* attribute), 521  
`has_setup` (*robot.running.model.If* attribute), 525  
`has_setup` (*robot.running.model.IfBranch* attribute), 524  
`has_setup` (*robot.running.model.Keyword* attribute), 519  
`has_setup` (*robot.running.model.Return* attribute), 529  
`has_setup` (*robot.running.model.TestCase* attribute), 533  
`has_setup` (*robot.running.model.TestSuite* attribute), 536  
`has_setup` (*robot.running.model.Try* attribute), 528  
`has_setup` (*robot.running.model.TryBranch* attribute), 526  
`has_setup` (*robot.running.model.While* attribute), 522  
`has_teardown` (*robot.model.body.BodyItem* attribute), 193  
`has_teardown` (*robot.model.control.Break* attribute), 215  
`has_teardown` (*robot.model.control.Continue* attribute), 214  
`has_teardown` (*robot.model.control.For* attribute), 205  
`has_teardown` (*robot.model.control.If* attribute), 209  
`has_teardown` (*robot.model.control.IfBranch* attribute), 207  
`has_teardown` (*robot.model.control.Return* attribute), 212  
`has_teardown` (*robot.model.control.Try* attribute), 211  
`has_teardown` (*robot.model.control.TryBranch* attribute), 210  
`has_teardown` (*robot.model.control.While* attribute), 206  
`has_teardown` (*robot.model.keyword.Keyword* attribute), 225  
`has_teardown` (*robot.model.message.Message* attribute), 229  
`has_teardown` (*robot.model.testcase.TestCase* attribute), 248  
`has_teardown` (*robot.model.testsuite.TestSuite* attribute), 251  
`has_teardown` (*robot.output.loggerhelper.Message* attribute), 275  
`has_teardown` (*robot.result.model.Break* attribute), 453  
`has_teardown` (*robot.result.model.Continue* attribute), 451  
`has_teardown` (*robot.result.model.For* attribute), 432  
`has_teardown` (*robot.result.model.ForIteration* attribute), 430  
`has_teardown` (*robot.result.model.If* attribute), 441  
`has_teardown` (*robot.result.model.IfBranch* attribute), 439  
`has_teardown` (*robot.result.model.Keyword* attribute), 456  
`has_teardown` (*robot.result.model.Message* attribute), 428  
`has_teardown` (*robot.result.model.Return* attribute), 448  
`has_teardown` (*robot.result.model.TestCase* attribute), 458  
`has_teardown` (*robot.result.model.TestSuite* attribute), 461  
`has_teardown` (*robot.result.model.Try* attribute), 446  
`has_teardown` (*robot.result.model.TryBranch* attribute), 444  
`has_teardown` (*robot.result.model.While* attribute), 437  
`has_teardown` (*robot.result.model.WhileIteration* attribute), 435  
`has_teardown` (*robot.running.model.Break* attribute), 532  
`has_teardown` (*robot.running.model.Continue* attribute), 531  
`has_teardown` (*robot.running.model.For* attribute), 521  
`has_teardown` (*robot.running.model.If* attribute), 525  
`has_teardown` (*robot.running.model.IfBranch* attribute), 524  
`has_teardown` (*robot.running.model.Keyword* attribute), 519  
`has_teardown` (*robot.running.model.Return* attribute), 529  
`has_teardown` (*robot.running.model.TestCase* attribute), 533  
`has_teardown` (*robot.running.model.TestSuite* attribute), 537  
`has_teardown` (*robot.running.model.Try* attribute), 528  
`has_teardown` (*robot.running.model.TryBranch* attribute), 526  
`has_teardown` (*robot.running.model.While* attribute), 522



- has\_tests (*robot.model.testsuite.TestSuite* attribute), 251
- has\_tests (*robot.result.model.TestSuite* attribute), 461
- has\_tests (*robot.running.model.TestSuite* attribute), 537
- HEADER\_TOKENS (*robot.parsing.lexer.tokens.END* attribute), 304
- HEADER\_TOKENS (*robot.parsing.lexer.tokens.EOS* attribute), 302
- HEADER\_TOKENS (*robot.parsing.lexer.tokens.Token* attribute), 301
- HeaderAndBody (class in *robot.parsing.model.blocks*), 306
- HeaderFormatter (class in *robot.utils.htmlformatters*), 561
- highlight() (*robot.output.console.highlighting.HighlightingStream* method), 268
- Highlighter() (in module *robot.output.console.highlighting*), 268
- HighlightingStream (class in *robot.output.console.highlighting*), 268
- html (*robot.model.message.Message* attribute), 227
- html (*robot.output.loggerhelper.Message* attribute), 275
- html (*robot.result.model.Message* attribute), 428
- html() (*robot.libdocpkg.htmlutils.DocFormatter* method), 36
- html() (*robot.reporting.jsbuildingcontext JsBuildingContext* method), 357
- html\_chars (*robot.libdocpkg.htmlutils.HtmlToText* attribute), 37
- html\_escape() (in module *robot.utils.markuputils*), 564
- html\_format() (in module *robot.utils.markuputils*), 564
- html\_message (*robot.model.message.Message* attribute), 227
- html\_message (*robot.output.loggerhelper.Message* attribute), 275
- html\_message (*robot.result.model.Message* attribute), 428
- html\_tags (*robot.libdocpkg.htmlutils.HtmlToText* attribute), 37
- html\_to\_plain\_text() (*robot.libdocpkg.htmlutils.HtmlToText* method), 37
- HtmlFileWriter (class in *robot.htmldata.htmlfilewriter*), 33
- HtmlFormatter (class in *robot.utils.htmlformatters*), 561
- HtmlTemplate (class in *robot.htmldata.template*), 35
- HtmlToText (class in *robot.libdocpkg.htmlutils*), 37
- HtmlWriter (class in *robot.utils.markupwriters*), 564
- iconbitmap() (*robot.libraries.dialogs\_py.InputDialog* method), 141
- iconbitmap() (*robot.libraries.dialogs\_py.MessageDialog* method), 127
- iconbitmap() (*robot.libraries.dialogs\_py.MultipleSelectionDialog* method), 169
- iconbitmap() (*robot.libraries.dialogs\_py.PassFailDialog* method), 183
- iconbitmap() (*robot.libraries.dialogs\_py.SelectionDialog* method), 155
- iconify() (*robot.libraries.dialogs\_py.InputDialog* method), 141
- iconify() (*robot.libraries.dialogs\_py.MessageDialog* method), 128
- iconify() (*robot.libraries.dialogs\_py.MultipleSelectionDialog* method), 169
- iconify() (*robot.libraries.dialogs\_py.PassFailDialog* method), 183
- iconify() (*robot.libraries.dialogs\_py.SelectionDialog* method), 155
- iconmask() (*robot.libraries.dialogs\_py.InputDialog* method), 142
- iconmask() (*robot.libraries.dialogs\_py.MessageDialog* method), 128
- iconmask() (*robot.libraries.dialogs\_py.MultipleSelectionDialog* method), 170
- iconmask() (*robot.libraries.dialogs\_py.PassFailDialog* method), 184
- iconmask() (*robot.libraries.dialogs\_py.SelectionDialog* method), 156
- iconname() (*robot.libraries.dialogs\_py.InputDialog* method), 142
- iconname() (*robot.libraries.dialogs\_py.MessageDialog* method), 128
- iconname() (*robot.libraries.dialogs\_py.MultipleSelectionDialog* method), 170
- iconname() (*robot.libraries.dialogs\_py.PassFailDialog* method), 184
- iconname() (*robot.libraries.dialogs\_py.SelectionDialog* method), 156
- iconphoto() (*robot.libraries.dialogs\_py.InputDialog* method), 142
- iconphoto() (*robot.libraries.dialogs\_py.MessageDialog* method), 128
- iconphoto() (*robot.libraries.dialogs\_py.MultipleSelectionDialog* method), 170
- iconphoto() (*robot.libraries.dialogs\_py.PassFailDialog* method), 184
- iconphoto() (*robot.libraries.dialogs\_py.SelectionDialog* method), 156
- iconposition() (*robot.libraries.dialogs\_py.InputDialog* method), 142
- iconposition() (*robot.libraries.dialogs\_py.MessageDialog* method), 128

- method*), 128
- `iconposition()` (*robot.libraries.dialogs\_py.MultipleSelectionDialog* attribute), 170
- method*), 184
- `iconposition()` (*robot.libraries.dialogs\_py.SelectionDialog* attribute), 156
- method*), 142
- `iconwindow()` (*robot.libraries.dialogs\_py.MessageDialog* attribute), 128
- method*), 170
- `iconwindow()` (*robot.libraries.dialogs\_py.PassFailDialog* attribute), 184
- method*), 156
- `id` (*robot.model.body.BodyItem* attribute), 193
- `id` (*robot.model.control.Break* attribute), 215
- `id` (*robot.model.control.Continue* attribute), 214
- `id` (*robot.model.control.For* attribute), 205
- `id` (*robot.model.control.If* attribute), 208
- `id` (*robot.model.control.IfBranch* attribute), 206
- `id` (*robot.model.control.Return* attribute), 212
- `id` (*robot.model.control.Try* attribute), 210
- `id` (*robot.model.control.TryBranch* attribute), 209
- `id` (*robot.model.control.While* attribute), 206
- `id` (*robot.model.keyword.Keyword* attribute), 226
- `id` (*robot.model.message.Message* attribute), 228
- `id` (*robot.model.stats.SuiteStat* attribute), 240
- `id` (*robot.model.testcase.TestCase* attribute), 248
- `id` (*robot.model.testsuite.TestSuite* attribute), 251
- `id` (*robot.output.loggerhelper.Message* attribute), 275
- `id` (*robot.result.executionerrors.ExecutionErrors* attribute), 376
- `id` (*robot.result.model.Break* attribute), 453
- `id` (*robot.result.model.Continue* attribute), 451
- `id` (*robot.result.model.For* attribute), 432
- `id` (*robot.result.model.ForIteration* attribute), 430
- `id` (*robot.result.model.If* attribute), 442
- `id` (*robot.result.model.IfBranch* attribute), 439
- `id` (*robot.result.model.Keyword* attribute), 456
- `id` (*robot.result.model.Message* attribute), 428
- `id` (*robot.result.model.Return* attribute), 448
- `id` (*robot.result.model.TestCase* attribute), 458
- `id` (*robot.result.model.TestSuite* attribute), 461
- `id` (*robot.result.model.Try* attribute), 446
- `id` (*robot.result.model.TryBranch* attribute), 444
- `id` (*robot.result.model.While* attribute), 437
- `id` (*robot.result.model.WhileIteration* attribute), 435
- `id` (*robot.running.model.Break* attribute), 532
- `id` (*robot.running.model.Continue* attribute), 531
- `id` (*robot.running.model.For* attribute), 521
- `id` (*robot.running.model.If* attribute), 525
- `id` (*robot.running.model.IfBranch* attribute), 524
- `id` (*robot.running.model.Keyword* attribute), 519
- `id` (*robot.running.model.Return* attribute), 529
- `id` (*robot.running.model.TestCase* attribute), 533
- `id` (*robot.running.model.TestSuite* attribute), 537
- `id` (*robot.running.model.Try* attribute), 528
- `id` (*robot.running.model.TryBranch* attribute), 527
- `id` (*robot.running.model.While* attribute), 522
- `identifiers` (*robot.variables.finders.EmptyFinder* attribute), 569
- `identifiers` (*robot.variables.finders.EnvironmentFinder* attribute), 569
- `identifiers` (*robot.variables.finders.ExtendedFinder* attribute), 569
- `identifiers` (*robot.variables.finders.InlinePythonFinder* attribute), 569
- `identifiers` (*robot.variables.finders.NumberFinder* attribute), 569
- `identifiers` (*robot.variables.finders.StoredFinder* attribute), 569
- `If` (class in *robot.model.control*), 207
- `If` (class in *robot.parsing.model.blocks*), 308
- `If` (class in *robot.result.model*), 440
- `If` (class in *robot.running.model*), 524
- `IF` (*robot.model.body.BodyItem* attribute), 193
- `IF` (*robot.model.control.Break* attribute), 214
- `IF` (*robot.model.control.Continue* attribute), 213
- `IF` (*robot.model.control.For* attribute), 204
- `IF` (*robot.model.control.If* attribute), 208
- `IF` (*robot.model.control.IfBranch* attribute), 207
- `IF` (*robot.model.control.Return* attribute), 212
- `IF` (*robot.model.control.Try* attribute), 211
- `IF` (*robot.model.control.TryBranch* attribute), 209
- `IF` (*robot.model.control.While* attribute), 205
- `IF` (*robot.model.keyword.Keyword* attribute), 225
- `IF` (*robot.model.message.Message* attribute), 228
- `IF` (*robot.output.loggerhelper.Message* attribute), 275
- `IF` (*robot.parsing.lexer.tokens.END* attribute), 304
- `IF` (*robot.parsing.lexer.tokens.EOS* attribute), 302
- `IF` (*robot.parsing.lexer.tokens.Token* attribute), 300
- `IF` (*robot.result.model.Break* attribute), 452
- `IF` (*robot.result.model.Continue* attribute), 450
- `IF` (*robot.result.model.For* attribute), 431
- `IF` (*robot.result.model.ForIteration* attribute), 429
- `IF` (*robot.result.model.If* attribute), 440
- `IF` (*robot.result.model.IfBranch* attribute), 438
- `IF` (*robot.result.model.Keyword* attribute), 455
- `IF` (*robot.result.model.Message* attribute), 427
- `IF` (*robot.result.model.Return* attribute), 447
- `IF` (*robot.result.model.Try* attribute), 445
- `IF` (*robot.result.model.TryBranch* attribute), 443
- `IF` (*robot.result.model.While* attribute), 436
- `IF` (*robot.result.model.WhileIteration* attribute), 434
- `IF` (*robot.running.model.Break* attribute), 531

- IF (*robot.running.model.Continue* attribute), 530
- IF (*robot.running.model.For* attribute), 520
- IF (*robot.running.model.If* attribute), 524
- IF (*robot.running.model.IfBranch* attribute), 523
- IF (*robot.running.model.Keyword* attribute), 518
- IF (*robot.running.model.Return* attribute), 529
- IF (*robot.running.model.Try* attribute), 527
- IF (*robot.running.model.TryBranch* attribute), 526
- IF (*robot.running.model.While* attribute), 522
- if\_class (*robot.model.body.BaseBody* attribute), 194
- if\_class (*robot.model.body.Body* attribute), 197
- if\_class (*robot.model.body.Branches* attribute), 198
- if\_class (*robot.result.model.Body* attribute), 423
- if\_class (*robot.result.model.Branches* attribute), 424
- if\_class (*robot.result.model.Iterations* attribute), 426
- if\_class (*robot.running.model.Body* attribute), 517
- IF\_ELSE\_ROOT (*robot.model.body.BodyItem* attribute), 193
- IF\_ELSE\_ROOT (*robot.model.control.Break* attribute), 214
- IF\_ELSE\_ROOT (*robot.model.control.Continue* attribute), 213
- IF\_ELSE\_ROOT (*robot.model.control.For* attribute), 204
- IF\_ELSE\_ROOT (*robot.model.control.If* attribute), 208
- IF\_ELSE\_ROOT (*robot.model.control.IfBranch* attribute), 207
- IF\_ELSE\_ROOT (*robot.model.control.Return* attribute), 212
- IF\_ELSE\_ROOT (*robot.model.control.Try* attribute), 211
- IF\_ELSE\_ROOT (*robot.model.control.TryBranch* attribute), 209
- IF\_ELSE\_ROOT (*robot.model.control.While* attribute), 205
- IF\_ELSE\_ROOT (*robot.model.keyword.Keyword* attribute), 225
- IF\_ELSE\_ROOT (*robot.model.message.Message* attribute), 228
- IF\_ELSE\_ROOT (*robot.output.loggerhelper.Message* attribute), 275
- IF\_ELSE\_ROOT (*robot.result.model.Break* attribute), 452
- IF\_ELSE\_ROOT (*robot.result.model.Continue* attribute), 450
- IF\_ELSE\_ROOT (*robot.result.model.For* attribute), 431
- IF\_ELSE\_ROOT (*robot.result.model.ForIteration* attribute), 429
- IF\_ELSE\_ROOT (*robot.result.model.If* attribute), 440
- IF\_ELSE\_ROOT (*robot.result.model.IfBranch* attribute), 438
- IF\_ELSE\_ROOT (*robot.result.model.Keyword* attribute), 455
- IF\_ELSE\_ROOT (*robot.result.model.Message* attribute), 427
- IF\_ELSE\_ROOT (*robot.result.model.Return* attribute), 447
- IF\_ELSE\_ROOT (*robot.result.model.Try* attribute), 445
- IF\_ELSE\_ROOT (*robot.result.model.TryBranch* attribute), 443
- IF\_ELSE\_ROOT (*robot.result.model.While* attribute), 436
- IF\_ELSE\_ROOT (*robot.result.model.WhileIteration* attribute), 434
- IF\_ELSE\_ROOT (*robot.running.model.Break* attribute), 531
- IF\_ELSE\_ROOT (*robot.running.model.Continue* attribute), 530
- IF\_ELSE\_ROOT (*robot.running.model.For* attribute), 520
- IF\_ELSE\_ROOT (*robot.running.model.If* attribute), 524
- IF\_ELSE\_ROOT (*robot.running.model.IfBranch* attribute), 523
- IF\_ELSE\_ROOT (*robot.running.model.Keyword* attribute), 518
- IF\_ELSE\_ROOT (*robot.running.model.Return* attribute), 529
- IF\_ELSE\_ROOT (*robot.running.model.Try* attribute), 527
- IF\_ELSE\_ROOT (*robot.running.model.TryBranch* attribute), 526
- IF\_ELSE\_ROOT (*robot.running.model.While* attribute), 522
- IfBranch (class in *robot.model.control*), 206
- IfBranch (class in *robot.result.model*), 438
- IfBranch (class in *robot.running.model*), 523
- IfBuilder (class in *robot.running.builder.transformers*), 509
- IfElseHeader (class in *robot.parsing.model.statements*), 337
- IfHandler (class in *robot.result.xmlélémenthandlers*), 485
- IfHeader (class in *robot.parsing.model.statements*), 338
- IfHeaderLexer (class in *robot.parsing.lexer.statementlexers*), 296
- IfLexer (class in *robot.parsing.lexer.blocklexers*), 287
- IfParser (class in *robot.parsing.parser.blockparsers*), 354
- IfRunner (class in *robot.running.bodyrunner*), 512
- ignored\_dirs (*robot.parsing.suitestructure.SuiteStructureBuilder* attribute), 356
- ignored\_prefixes (*robot.parsing.suitestructure.SuiteStructureBuilder* attribute), 356
- imag (*robot.reporting.stringcache.StringIndex* attribute), 365
- image\_names () (*robot.libraries.dialogs\_py.InputDialog* method), 142

`image_names()` (*robot.libraries.dialogs\_py.MessageDialog* method), 128  
`image_names()` (*robot.libraries.dialogs\_py.MultipleSelectionDialog* method), 170  
`image_names()` (*robot.libraries.dialogs\_py.PassFailDialog* method), 184  
`image_names()` (*robot.libraries.dialogs\_py.SelectionDialog* method), 156  
`image_types()` (*robot.libraries.dialogs\_py.InputDialog* method), 142  
`image_types()` (*robot.libraries.dialogs\_py.MessageDialog* method), 128  
`image_types()` (*robot.libraries.dialogs\_py.MultipleSelectionDialog* method), 170  
`image_types()` (*robot.libraries.dialogs\_py.PassFailDialog* method), 184  
`image_types()` (*robot.libraries.dialogs\_py.SelectionDialog* method), 156  
`ImplicitCommentSectionLexer` (class in *robot.parsing.lexer.blocklexers*), 285  
`ImplicitCommentSectionParser` (class in *robot.parsing.parser.fileparser*), 355  
`Import` (class in *robot.running.model*), 539  
`import_()` (*robot.utils.importer.ByPathImporter* method), 563  
`import_()` (*robot.utils.importer.DottedImporter* method), 564  
`import_()` (*robot.utils.importer.NonDottedImporter* method), 563  
`import_class_or_module()` (*robot.utils.importer.Importer* method), 563  
`import_class_or_module_by_path()` (*robot.utils.importer.Importer* method), 563  
`import_library()` (*robot.libraries.BuiltIn.BuiltIn* method), 48  
`import_library()` (*robot.running.importer.Importer* method), 514  
`import_library()` (*robot.running.namespace.Namespace* method), 540  
`import_listeners()` (*robot.output.listeners.ListenerProxy* class method), 272  
`import_resource()` (*robot.libraries.BuiltIn.BuiltIn* method), 48  
`import_resource()` (*robot.running.importer.Importer* method), 514  
`import_resource()` (*robot.running.namespace.Namespace* method), 540  
`import_variables()` (*robot.libraries.BuiltIn.BuiltIn* method), 48  
`import_variables()` (*robot.running.namespace.Namespace* method), 540  
`import_variables()` (*robot.variables.filesetter.PythonImporter* method), 569  
`import_variables()` (*robot.variables.filesetter.YamlImporter* method), 569  
`ImportCache` (class in *robot.running.importer*), 514  
`imported()` (*robot.output.listeners.LibraryListeners* method), 272  
`imported()` (*robot.output.listeners.Listeners* method), 272  
`imported()` (*robot.output.logger.Logger* method), 273  
`Importer` (class in *robot.running.importer*), 514  
`Importer` (class in *robot.utils.importer*), 562  
`Imports` (class in *robot.running.model*), 539  
`imports` (*robot.running.model.ResourceFile* attribute), 538  
`in_for` (*robot.parsing.model.blocks.ValidationContext* attribute), 310  
`in_keyword` (*robot.parsing.model.blocks.ValidationContext* attribute), 310  
`in_while` (*robot.parsing.model.blocks.ValidationContext* attribute), 310  
`include` (*robot.conf.settings.RebotSettings* attribute), 33  
`include` (*robot.conf.settings.RobotSettings* attribute), 32  
`include_suites` (*robot.model.filter.Filter* attribute), 219  
`include_tags` (*robot.model.filter.Filter* attribute), 219  
`include_tests` (*robot.model.filter.Filter* attribute), 219  
`index()` (*robot.model.body.BaseBody* method), 195  
`index()` (*robot.model.body.Body* method), 197  
`index()` (*robot.model.body.Branches* method), 198  
`index()` (*robot.model.itemlist.ItemList* method), 224  
`index()` (*robot.model.keyword.Keywords* method), 227  
`index()` (*robot.model.message.Messages* method), 229  
`index()` (*robot.model.testcase.TestCases* method), 249  
`index()` (*robot.model.testsuite.TestSuites* method), 253  
`index()` (*robot.result.model.Body* method), 423  
`index()` (*robot.result.model.Branches* method), 424  
`index()` (*robot.result.model.Iterations* method), 426  
`index()` (*robot.running.model.Body* method), 517  
`index()` (*robot.running.model.Imports* method), 539  
`info` (*robot.model.stats.CombinedTagStat* attribute), 241  
`info` (*robot.model.stats.TagStat* attribute), 241  
`info()` (in module *robot.api.logger*), 15  
`info()` (in module *robot.output.librarylogger*), 270  
`info()` (*robot.output.console.verbose.VerboseWriter* method), 269



`info()` (*robot.output.filelogger.FileLogger* method), 270  
`info()` (*robot.output.logger.Logger* method), 274  
`info()` (*robot.output.loggerhelper.AbstractLogger* method), 274  
`info()` (*robot.output.output.Output* method), 276  
`info()` (*robot.utils.application.DefaultLogger* method), 554  
`info()` (*robot.utils.importer.NoLogger* method), 564  
Information, 576  
`InitFileContext` (class *in robot.parsing.lexer.context*), 289  
`InitFileSections` (class *in robot.parsing.lexer.sections*), 291  
`InitFileSettings` (class *in robot.parsing.lexer.settings*), 292  
`InitHandler()` (*in module robot.running.handlers*), 513  
`inits` (*robot.libdocpkg.model.LibraryDoc* attribute), 37  
`INLINE_IF` (*robot.parsing.lexer.tokens.END* attribute), 304  
`INLINE_IF` (*robot.parsing.lexer.tokens.EOS* attribute), 302  
`INLINE_IF` (*robot.parsing.lexer.tokens.Token* attribute), 300  
`InlineIfHeader` (class *in robot.parsing.model.statements*), 339  
`InlineIfHeaderLexer` (class *in robot.parsing.lexer.statementslexers*), 297  
`InlineIfLexer` (class *in robot.parsing.lexer.blocklexers*), 287  
`InlinePythonFinder` (class *in robot.variables.finders*), 569  
`input()` (*robot.parsing.lexer.blocklexers.BlockLexer* method), 283  
`input()` (*robot.parsing.lexer.blocklexers.CommentSectionLexer* method), 285  
`input()` (*robot.parsing.lexer.blocklexers.ErrorSectionLexer* method), 286  
`input()` (*robot.parsing.lexer.blocklexers.FileLexer* method), 284  
`input()` (*robot.parsing.lexer.blocklexers.ForLexer* method), 287  
`input()` (*robot.parsing.lexer.blocklexers.IfLexer* method), 287  
`input()` (*robot.parsing.lexer.blocklexers.ImplicitCommentSectionLexer* method), 285  
`input()` (*robot.parsing.lexer.blocklexers.InlineIfLexer* method), 287  
`input()` (*robot.parsing.lexer.blocklexers.KeywordLexer* method), 286  
`input()` (*robot.parsing.lexer.blocklexers.KeywordSectionLexer* method), 285  
`input()` (*robot.parsing.lexer.blocklexers.NestedBlockLexer* method), 287  
`input()` (*robot.parsing.lexer.blocklexers.SectionLexer* method), 284  
`input()` (*robot.parsing.lexer.blocklexers.SettingSectionLexer* method), 284  
`input()` (*robot.parsing.lexer.blocklexers.TestCaseLexer* method), 286  
`input()` (*robot.parsing.lexer.blocklexers.TestCaseSectionLexer* method), 285  
`input()` (*robot.parsing.lexer.blocklexers.TestOrKeywordLexer* method), 286  
`input()` (*robot.parsing.lexer.blocklexers.TryLexer* method), 288  
`input()` (*robot.parsing.lexer.blocklexers.VariableSectionLexer* method), 284  
`input()` (*robot.parsing.lexer.blocklexers.WhileLexer* method), 287  
`input()` (*robot.parsing.lexer.lexer.Lexer* method), 290  
`input()` (*robot.parsing.lexer.statementslexers.BreakLexer* method), 299  
`input()` (*robot.parsing.lexer.statementslexers.CommentLexer* method), 295  
`input()` (*robot.parsing.lexer.statementslexers.CommentSectionHeaderLexer* method), 295  
`input()` (*robot.parsing.lexer.statementslexers.ContinueLexer* method), 298  
`input()` (*robot.parsing.lexer.statementslexers.ElseHeaderLexer* method), 297  
`input()` (*robot.parsing.lexer.statementslexers.ElseIfHeaderLexer* method), 297  
`input()` (*robot.parsing.lexer.statementslexers.EndLexer* method), 298  
`input()` (*robot.parsing.lexer.statementslexers.ErrorSectionHeaderLexer* method), 295  
`input()` (*robot.parsing.lexer.statementslexers.ExceptHeaderLexer* method), 298  
`input()` (*robot.parsing.lexer.statementslexers.FinallyHeaderLexer* method), 298  
`input()` (*robot.parsing.lexer.statementslexers.ForHeaderLexer* method), 296  
`input()` (*robot.parsing.lexer.statementslexers.IfHeaderLexer* method), 297  
`input()` (*robot.parsing.lexer.statementslexers.InlineIfHeaderLexer* method), 297  
`input()` (*robot.parsing.lexer.statementslexers.KeywordCallLexer* method), 296  
`input()` (*robot.parsing.lexer.statementslexers.KeywordSectionHeaderLexer* method), 295  
`input()` (*robot.parsing.lexer.statementslexers.Lexer* method), 293  
`input()` (*robot.parsing.lexer.statementslexers.ReturnLexer* method), 298  
`input()` (*robot.parsing.lexer.statementslexers.SectionHeaderLexer* method), 294

() (*robot.parsing.lexer.statemntlexers.SettingLexerInvalidLimit* (class in *robot.running.bodyrunner*), method), 296  
() (*robot.parsing.lexer.statemntlexers.SettingSectionHeaderLexer* (in module *robot.variables.search*), method), 294  
() (*robot.parsing.lexer.statemntlexers.SingleType* method), 294  
() (*robot.parsing.lexer.statemntlexers.StatementLexer* method), 294  
() (*robot.parsing.lexer.statemntlexers.TestCaseSectionHeaderLexer* method), 295  
() (*robot.parsing.lexer.statemntlexers.TestOrKeywordSettingLexer* method), 296  
() (*robot.parsing.lexer.statemntlexers.TryHeaderLexer* method), 297  
() (*robot.parsing.lexer.statemntlexers.TypeAndArguments* method), 294  
() (*robot.parsing.lexer.statemntlexers.VariableLexer* method), 296  
() (*robot.parsing.lexer.statemntlexers.VariableSectionHeaderLexer* method), 295  
() (*robot.parsing.lexer.statemntlexers.WhileHeaderLexer* method), 298  
InputDialog (class in *robot.libraries.dialogs\_py*), 136  
insert() (*robot.model.body.BaseBody* method), 195  
insert() (*robot.model.body.Body* method), 197  
insert() (*robot.model.body.Branches* method), 198  
insert() (*robot.model.itemlist.ItemList* method), 224  
insert() (*robot.model.keyword.Keywords* method), 227  
insert() (*robot.model.message.Messages* method), 229  
insert() (*robot.model.testcase.TestCases* method), 249  
insert() (*robot.model.testsuite.TestSuites* method), 253  
insert() (*robot.result.model.Body* method), 423  
insert() (*robot.result.model.Branches* method), 424  
insert() (*robot.result.model.Iterations* method), 426  
insert() (*robot.running.model.Body* method), 517  
insert() (*robot.running.model.Imports* method), 539  
insert\_into\_list() (*robot.libraries.Collections.Collections* method), 69  
IntegerConverter (class in *robot.running.arguments.typeconverters*), 497  
IntegerDumper (class in *robot.htmldata.jsonwriter*), 34  
interact() (*robot.libraries.Telnet.TelnetConnection* method), 110  
INTERNAL\_UPDATE\_FREQUENCY (*robot.libraries.Telnet.TelnetConnection* attribute), 107  
is\_assign() (*robot.variables.search.VariableMatch* method), 572  
is\_dict\_assign() (in module *robot.variables.search*), 572  
is\_dict\_assign() (*robot.variables.search.VariableMatch* method), 572  
is\_dict\_variable() (in module *robot.variables.search*), 572  
is\_dict\_variable() (*robot.variables.search.VariableMatch* method), 572  
is\_in\_directory(*robot.parsing.suitestructure.SuiteStructure* attribute), 356  
is\_global(*robot.running.libraryscopes.GlobalScope* attribute), 515  
is\_global(*robot.running.libraryscopes.TestCaseScope* attribute), 516  
is\_global(*robot.running.libraryscopes.TestSuiteScope* attribute), 515  
is\_list\_assign() (in module *robot.variables.search*), 572  
is\_list\_assign() (*robot.variables.search.VariableMatch* method), 572  
is\_list\_variable() (in module *robot.variables.search*), 572  
is\_list\_variable() (*robot.variables.search.VariableMatch* method), 572  
is\_process\_running() (*robot.libraries.Process.Process* method), 91  
is\_scalar\_assign() (in module *robot.variables.search*), 572  
is\_scalar\_assign() (*robot.variables.search.VariableMatch* method), 572  
is\_scalar\_variable() (in module *robot.variables.search*), 572  
is\_scalar\_variable() (*robot.variables.search.VariableMatch* method), 572  
is\_valid(*robot.running.bodyrunner.DurationLimit* attribute), 512  
is\_valid(*robot.running.bodyrunner.InvalidLimit* attribute), 512  
is\_valid(*robot.running.bodyrunner.IterationCountLimit* attribute), 512  
is\_valid(*robot.running.bodyrunner.NoLimit* attribute), 512  
is\_valid(*robot.running.bodyrunner.WhileLimit* attribute), 512

- [is\\_variable\(\) \(in module robot.variables.search\), 572](#)  
[is\\_variable\(\) \(robot.variables.search.VariableMatch method\), 572](#)  
[isatty\(\) \(in module robot.utils.misc\), 566](#)  
[isatty\(\) \(in module robot.utils.platform\), 566](#)  
[IsLogged \(class in robot.output.loggerhelper\), 276](#)  
[item\\_state\(\) \(robot.variables.search.VariableSearcher method\), 573](#)  
[ItemList \(class in robot.model.itemlist\), 224](#)  
[items\(\) \(robot.model.metadata.Metadata method\), 229](#)  
[items\(\) \(robot.utils.dotdict.DotDict method\), 559](#)  
[items\(\) \(robot.utils.normalizing.NormalizedDict method\), 566](#)  
[items\(\) \(robot.variables.evaluation.EvaluationNamespace method\), 568](#)  
[ITERATION \(robot.model.body.BodyItem attribute\), 193](#)  
[ITERATION \(robot.model.control.Break attribute\), 214](#)  
[ITERATION \(robot.model.control.Continue attribute\), 213](#)  
[ITERATION \(robot.model.control.For attribute\), 204](#)  
[ITERATION \(robot.model.control.If attribute\), 208](#)  
[ITERATION \(robot.model.control.IfBranch attribute\), 207](#)  
[ITERATION \(robot.model.control.Return attribute\), 212](#)  
[ITERATION \(robot.model.control.Try attribute\), 211](#)  
[ITERATION \(robot.model.control.TryBranch attribute\), 209](#)  
[ITERATION \(robot.model.control.While attribute\), 205](#)  
[ITERATION \(robot.model.keyword.Keyword attribute\), 225](#)  
[ITERATION \(robot.model.message.Message attribute\), 228](#)  
[ITERATION \(robot.output.loggerhelper.Message attribute\), 275](#)  
[ITERATION \(robot.result.model.Break attribute\), 452](#)  
[ITERATION \(robot.result.model.Continue attribute\), 450](#)  
[ITERATION \(robot.result.model.For attribute\), 431](#)  
[ITERATION \(robot.result.model.ForIteration attribute\), 429](#)  
[ITERATION \(robot.result.model.If attribute\), 440](#)  
[ITERATION \(robot.result.model.IfBranch attribute\), 438](#)  
[ITERATION \(robot.result.model.Keyword attribute\), 455](#)  
[ITERATION \(robot.result.model.Message attribute\), 427](#)  
[ITERATION \(robot.result.model.Return attribute\), 447](#)  
[ITERATION \(robot.result.model.Try attribute\), 445](#)  
[ITERATION \(robot.result.model.TryBranch attribute\), 443](#)  
[ITERATION \(robot.result.model.While attribute\), 436](#)  
[ITERATION \(robot.result.model.WhileIteration attribute\), 434](#)  
[ITERATION \(robot.running.model.Break attribute\), 531](#)  
[ITERATION \(robot.running.model.Continue attribute\), 530](#)  
[ITERATION \(robot.running.model.For attribute\), 520](#)  
[ITERATION \(robot.running.model.If attribute\), 524](#)  
[ITERATION \(robot.running.model.IfBranch attribute\), 523](#)  
[ITERATION \(robot.running.model.Keyword attribute\), 518](#)  
[ITERATION \(robot.running.model.Return attribute\), 529](#)  
[ITERATION \(robot.running.model.Try attribute\), 527](#)  
[ITERATION \(robot.running.model.TryBranch attribute\), 526](#)  
[ITERATION \(robot.running.model.While attribute\), 522](#)  
[iteration\\_class \(robot.result.model.For attribute\), 431](#)  
[iteration\\_class \(robot.result.model.Iterations attribute\), 425](#)  
[iteration\\_class \(robot.result.model.While attribute\), 435](#)  
[IterationCountLimit \(class in robot.running.bodyrunner\), 512](#)  
[IterationHandler \(class in robot.result.xmllelementhandlers\), 485](#)  
[Iterations \(class in robot.result.model\), 425](#)  
[iterations\\_class \(robot.result.model.For attribute\), 431](#)  
[iterations\\_class \(robot.result.model.While attribute\), 435](#)
- ## J
- [join\\_command\\_line\(\) \(robot.libraries.Process.Process method\), 93](#)  
[join\\_path\(\) \(robot.libraries.OperatingSystem.OperatingSystem method\), 85](#)  
[join\\_paths\(\) \(robot.libraries.OperatingSystem.OperatingSystem method\), 85](#)  
[js\\_result \(robot.reporting.resultwriter.Results attribute\), 365](#)  
[JsBuildContext \(class in robot.reporting.jsbuildingcontext\), 357](#)  
[JsExecutionResult \(class in robot.reporting.jsexecutionresult\), 358](#)  
[JsFileWriter \(class in robot.htmldata.htmlfilewriter\), 34](#)  
[JsModelBuilder \(class in robot.reporting.jsmodelbuilders\), 358](#)  
[JsonConverter \(class in robot.testdoc\), 585](#)  
[JsonDocBuilder \(class in robot.libdocpkg.jsonbuilder\), 37](#)  
[JsonDumper \(class in robot.htmldata.jsonwriter\), 34](#)  
[JsonWriter \(class in robot.htmldata.jsonwriter\), 34](#)

`JsResultWriter` (class in `robot.reporting.jswriter`), 359

## K

`keep_in_dictionary()`  
(`robot.libraries.Collections.Collections` method), 70

`keys()` (`robot.libraries.dialogs_py.InputDialog` method), 142

`keys()` (`robot.libraries.dialogs_py.MessageDialog` method), 128

`keys()` (`robot.libraries.dialogs_py.MultipleSelectionDialog` method), 170

`keys()` (`robot.libraries.dialogs_py.PassFailDialog` method), 184

`keys()` (`robot.libraries.dialogs_py.SelectionDialog` method), 156

`keys()` (`robot.model.metadata.Metadata` method), 229

`keys()` (`robot.utils.dotdict.DotDict` method), 559

`keys()` (`robot.utils.normalizing.NormalizedDict` method), 566

`keys()` (`robot.variables.evaluation.EvaluationNamespace` method), 568

`Keyword` (class in `robot.model.keyword`), 224

`Keyword` (class in `robot.parsing.model.blocks`), 308

`Keyword` (class in `robot.result.model`), 454

`Keyword` (class in `robot.running.model`), 518

`KEYWORD` (`robot.model.body.BodyItem` attribute), 193

`KEYWORD` (`robot.model.control.Break` attribute), 214

`KEYWORD` (`robot.model.control.Continue` attribute), 213

`KEYWORD` (`robot.model.control.For` attribute), 204

`KEYWORD` (`robot.model.control.If` attribute), 208

`KEYWORD` (`robot.model.control.IfBranch` attribute), 207

`KEYWORD` (`robot.model.control.Return` attribute), 212

`KEYWORD` (`robot.model.control.Try` attribute), 211

`KEYWORD` (`robot.model.control.TryBranch` attribute), 209

`KEYWORD` (`robot.model.control.While` attribute), 205

`KEYWORD` (`robot.model.keyword.Keyword` attribute), 225

`KEYWORD` (`robot.model.message.Message` attribute), 228

`KEYWORD` (`robot.output.loggerhelper.Message` attribute), 275

`KEYWORD` (`robot.parsing.lexer.tokens.END` attribute), 304

`KEYWORD` (`robot.parsing.lexer.tokens.EOS` attribute), 302

`KEYWORD` (`robot.parsing.lexer.tokens.Token` attribute), 300

`keyword` (`robot.parsing.model.statements.KeywordCall` attribute), 335

`KEYWORD` (`robot.result.model.Break` attribute), 452

`KEYWORD` (`robot.result.model.Continue` attribute), 450

`KEYWORD` (`robot.result.model.For` attribute), 431

`KEYWORD` (`robot.result.model.ForIteration` attribute), 429

`KEYWORD` (`robot.result.model.If` attribute), 441

`KEYWORD` (`robot.result.model.IfBranch` attribute), 438

`KEYWORD` (`robot.result.model.Keyword` attribute), 455

`KEYWORD` (`robot.result.model.Message` attribute), 427

`KEYWORD` (`robot.result.model.Return` attribute), 447

`KEYWORD` (`robot.result.model.Try` attribute), 445

`KEYWORD` (`robot.result.model.TryBranch` attribute), 443

`KEYWORD` (`robot.result.model.While` attribute), 436

`KEYWORD` (`robot.result.model.WhileIteration` attribute), 434

`KEYWORD` (`robot.running.model.Break` attribute), 531

`KEYWORD` (`robot.running.model.Continue` attribute), 530

`KEYWORD` (`robot.running.model.For` attribute), 520

`KEYWORD` (`robot.running.model.If` attribute), 524

`KEYWORD` (`robot.running.model.IfBranch` attribute), 523

`KEYWORD` (`robot.running.model.Keyword` attribute), 518

`KEYWORD` (`robot.running.model.Return` attribute), 529

`KEYWORD` (`robot.running.model.Try` attribute), 527

`KEYWORD` (`robot.running.model.TryBranch` attribute), 526

`KEYWORD` (`robot.running.model.While` attribute), 522

`keyword()` (in module `robot.api.deco`), 11

`keyword()` (`robot.parsing.lexer.sections.InitFileSections` method), 291

`keyword()` (`robot.parsing.lexer.sections.ResourceFileSections` method), 291

`keyword()` (`robot.parsing.lexer.sections.Sections` method), 291

`keyword()` (`robot.parsing.lexer.sections.TestCaseFileSections` method), 291

`keyword_class` (`robot.model.body.BaseBody` attribute), 194

`keyword_class` (`robot.model.body.Body` attribute), 197

`keyword_class` (`robot.model.body.Branches` attribute), 198

`keyword_class` (`robot.result.model.Body` attribute), 423

`keyword_class` (`robot.result.model.Branches` attribute), 424

`keyword_class` (`robot.result.model.Iterations` attribute), 426

`keyword_class` (`robot.running.model.Body` attribute), 517

`keyword_context()`  
(`robot.parsing.lexer.context.FileContext` method), 288

`keyword_context()`  
(`robot.parsing.lexer.context.InitFileContext` method), 289

`keyword_context()`  
(`robot.parsing.lexer.context.ResourceFileContext`



<i>method</i> ), 289	KeywordContext (class in <i>robot.parsing.lexer.context</i> ), 289	in
<code>keyword_context()</code> ( <i>robot.parsing.lexer.context.TestCaseFileContext</i> <i>method</i> ), 288	KeywordDoc (class in <i>robot.libdocpkg.model</i> ), 38	
KEYWORD_HEADER ( <i>robot.parsing.lexer.tokens.END</i> attribute), 304	KeywordDocBuilder (class in <i>robot.libdocpkg.robotbuilder</i> ), 38	in
KEYWORD_HEADER ( <i>robot.parsing.lexer.tokens.EOS</i> attribute), 302	KeywordError, 575	
KEYWORD_HEADER ( <i>robot.parsing.lexer.tokens.Token</i> attribute), 299	KeywordHandler (class in <i>robot.result.xmllelementhandlers</i> ), 484	in
<code>keyword_marker()</code> ( <i>robot.output.console.verbose.VerboseWriter</i> <i>method</i> ), 269	KeywordLexer (class in <i>robot.parsing.lexer.blocklexers</i> ), 286	in
<code>keyword_markers()</code> ( <i>robot.parsing.lexer.sections.InitFileSections</i> <i>method</i> ), 292	KeywordMarker (class in <i>robot.output.console.verbose</i> ), 269	in
<code>keyword_markers()</code> ( <i>robot.parsing.lexer.sections.ResourceFileSections</i> <i>method</i> ), 291	KeywordMatcher (class in <i>robot.libdocpkg.consoleviewer</i> ), 36	in
<code>keyword_markers()</code> ( <i>robot.parsing.lexer.sections.Sections</i> <i>method</i> ), 290	KeywordName (class in <i>robot.parsing.model.statements</i> ), 328	in
<code>keyword_markers()</code> ( <i>robot.parsing.lexer.sections.TestCaseFileSections</i> <i>method</i> ), 291	KeywordParser (class in <i>robot.parsing.parser.blockparsers</i> ), 354	in
KEYWORD_NAME ( <i>robot.parsing.lexer.tokens.END</i> attribute), 304	KeywordRecommendationFinder (class in <i>robot.running.namespace</i> ), 540	in
KEYWORD_NAME ( <i>robot.parsing.lexer.tokens.EOS</i> attribute), 302	KeywordRemover () (in <i>robot.result.keywordremover</i> ), 379	module
KEYWORD_NAME ( <i>robot.parsing.lexer.tokens.Token</i> attribute), 299	KeywordRunner (class in <i>robot.running.bodyrunner</i> ), 511	
<code>keyword_section()</code> ( <i>robot.parsing.lexer.context.FileContext</i> <i>method</i> ), 288	Keywords (class in <i>robot.model.keyword</i> ), 226	
<code>keyword_section()</code> ( <i>robot.parsing.lexer.context.InitFileContext</i> <i>method</i> ), 289	keywords ( <i>robot.libdocpkg.model.LibraryDoc</i> attribute), 37	
<code>keyword_section()</code> ( <i>robot.parsing.lexer.context.ResourceFileContext</i> <i>method</i> ), 289	keywords ( <i>robot.model.control.For</i> attribute), 204	
<code>keyword_section()</code> ( <i>robot.parsing.lexer.context.TestCaseFileContext</i> <i>method</i> ), 288	keywords ( <i>robot.model.testcase.TestCase</i> attribute), 248	
<code>keyword_should_exist()</code> ( <i>robot.libraries.BuiltIn.BuiltIn</i> <i>method</i> ), 48	keywords ( <i>robot.model.testsuite.TestSuite</i> attribute), 251	
<code>keyword_timeout</code> ( <i>robot.errors.TimeoutError</i> attribute), 575	keywords ( <i>robot.result.model.For</i> attribute), 432	
KeywordBuilder (class in <i>robot.reporting.jsmodelbuilders</i> ), 358	keywords ( <i>robot.result.model.Keyword</i> attribute), 454	
KeywordBuilder (class in <i>robot.running.builder.transformers</i> ), 508	keywords ( <i>robot.result.model.TestCase</i> attribute), 458	
KeywordCall (class in <i>robot.parsing.model.statements</i> ), 335	keywords ( <i>robot.result.model.TestSuite</i> attribute), 461	
KeywordCallLexer (class in <i>robot.parsing.lexer.statementlexers</i> ), 296	keywords ( <i>robot.running.model.For</i> attribute), 521	
KeywordCallTemplate (class in <i>robot.running.arguments.argumentmapper</i> ), 492	keywords ( <i>robot.running.model.ResourceFile</i> attribute), 538	
	keywords ( <i>robot.running.model.TestCase</i> attribute), 533	
	keywords ( <i>robot.running.model.TestSuite</i> attribute), 537	
	keywords ( <i>robot.running.model.UserKeyword</i> attribute), 538	
	KeywordSection (class in <i>robot.parsing.model.blocks</i> ), 307	in
	KeywordSectionHeaderLexer (class in <i>robot.parsing.lexer.statementlexers</i> ), 295	in
	KeywordSectionLexer (class in <i>robot.parsing.lexer.blocklexers</i> ), 285	in
	KeywordSectionParser (class in <i>robot.parsing.parser.fileparser</i> ), 355	in
	KeywordSettings (class in <i>robot.parsing.lexer.context</i> ), 289	in

- robot.parsing.lexer.settings*), 293
- KeywordStore (class in *robot.running.namespace*), 540
- KeywordTimeout (class in *robot.running.timeouts*), 510
- KILL\_TIMEOUT (*robot.libraries.Process.Process* attribute), 90
- kwname (*robot.result.model.Break* attribute), 453
- kwname (*robot.result.model.Continue* attribute), 451
- kwname (*robot.result.model.For* attribute), 432
- kwname (*robot.result.model.ForIteration* attribute), 430
- kwname (*robot.result.model.If* attribute), 442
- kwname (*robot.result.model.IfBranch* attribute), 439
- kwname (*robot.result.model.Keyword* attribute), 454
- kwname (*robot.result.model.Return* attribute), 448
- kwname (*robot.result.model.Try* attribute), 446
- kwname (*robot.result.model.TryBranch* attribute), 444
- kwname (*robot.result.model.While* attribute), 437
- kwname (*robot.result.model.WhileIteration* attribute), 435
- kwname (*robot.result.model.deprecation.DeprecatedAttributeMixin* attribute), 464
- ## L
- LastStatementFinder (class in *robot.parsing.model.blocks*), 311
- length\_should\_be () (*robot.libraries.BuiltIn.BuiltIn* method), 48
- level (*robot.model.message.Message* attribute), 227
- level (*robot.output.loggerhelper.Message* attribute), 275
- level (*robot.result.model.Message* attribute), 428
- lex () (*robot.parsing.lexer.blocklexers.BlockLexer* method), 283
- lex () (*robot.parsing.lexer.blocklexers.CommentSectionLexer* method), 285
- lex () (*robot.parsing.lexer.blocklexers.ErrorSectionLexer* method), 286
- lex () (*robot.parsing.lexer.blocklexers.FileLexer* method), 284
- lex () (*robot.parsing.lexer.blocklexers.ForLexer* method), 287
- lex () (*robot.parsing.lexer.blocklexers.IfLexer* method), 287
- lex () (*robot.parsing.lexer.blocklexers.ImplicitCommentSectionLexer* method), 285
- lex () (*robot.parsing.lexer.blocklexers.InlineIfLexer* method), 287
- lex () (*robot.parsing.lexer.blocklexers.KeywordLexer* method), 286
- lex () (*robot.parsing.lexer.blocklexers.KeywordSectionLexer* method), 285
- lex () (*robot.parsing.lexer.blocklexers.NestedBlockLexer* method), 287
- lex () (*robot.parsing.lexer.blocklexers.SectionLexer* method), 284
- lex () (*robot.parsing.lexer.blocklexers.SettingSectionLexer* method), 284
- lex () (*robot.parsing.lexer.blocklexers.TestCaseLexer* method), 286
- lex () (*robot.parsing.lexer.blocklexers.TestCaseSectionLexer* method), 285
- lex () (*robot.parsing.lexer.blocklexers.TestOrKeywordLexer* method), 286
- lex () (*robot.parsing.lexer.blocklexers.TryLexer* method), 288
- lex () (*robot.parsing.lexer.blocklexers.VariableSectionLexer* method), 284
- lex () (*robot.parsing.lexer.blocklexers.WhileLexer* method), 287
- lex () (*robot.parsing.lexer.settings.InitFileSettings* method), 292
- lex () (*robot.parsing.lexer.settings.KeywordSettings* method), 293
- lex () (*robot.parsing.lexer.settings.ResourceFileSettings* method), 293
- lex () (*robot.parsing.lexer.settings.Settings* method), 292
- lex () (*robot.parsing.lexer.settings.TestCaseFileSettings* method), 292
- lex () (*robot.parsing.lexer.settings.TestCaseSettings* method), 293
- lex () (*robot.parsing.lexer.statementlexers.BreakLexer* method), 299
- lex () (*robot.parsing.lexer.statementlexers.CommentLexer* method), 296
- lex () (*robot.parsing.lexer.statementlexers.CommentSectionHeaderLexer* method), 295
- lex () (*robot.parsing.lexer.statementlexers.ContinueLexer* method), 298
- lex () (*robot.parsing.lexer.statementlexers.ElseHeaderLexer* method), 297
- lex () (*robot.parsing.lexer.statementlexers.ElseIfHeaderLexer* method), 297
- lex () (*robot.parsing.lexer.statementlexers.EndLexer* method), 298
- lex () (*robot.parsing.lexer.statementlexers.ErrorSectionHeaderLexer* method), 295
- lex () (*robot.parsing.lexer.statementlexers.ExceptHeaderLexer* method), 298
- lex () (*robot.parsing.lexer.statementlexers.FinallyHeaderLexer* method), 298
- lex () (*robot.parsing.lexer.statementlexers.ForHeaderLexer* method), 296
- lex () (*robot.parsing.lexer.statementlexers.IfHeaderLexer* method), 297

`lex()` (`robot.parsing.lexer.statemlexers.InlineIfHeaderLexer` `lex_setting()` (`robot.parsing.lexer.context.FileContext` `method`), 297 `method`), 288  
`lex()` (`robot.parsing.lexer.statemlexers.KeywordCallLexer` `lex_setting()` (`robot.parsing.lexer.context.InitFileContext` `method`), 296 `method`), 289  
`lex()` (`robot.parsing.lexer.statemlexers.KeywordSectionHeaderLexer` `lex_setting()` (`robot.parsing.lexer.context.KeywordContext` `method`), 295 `method`), 290  
`lex()` (`robot.parsing.lexer.statemlexers.Lexer` `lex_setting()` (`robot.parsing.lexer.context.LexingContext` `method`), 293 `method`), 288  
`lex()` (`robot.parsing.lexer.statemlexers.ReturnLexer` `lex_setting()` (`robot.parsing.lexer.context.ResourceFileContext` `method`), 298 `method`), 289  
`lex()` (`robot.parsing.lexer.statemlexers.SectionHeaderLexer` `lex_setting()` (`robot.parsing.lexer.context.TestCaseContext` `method`), 294 `method`), 289  
`lex()` (`robot.parsing.lexer.statemlexers.SettingLexer` `lex_setting()` (`robot.parsing.lexer.context.TestCaseFileContext` `method`), 296 `method`), 289  
`lex()` (`robot.parsing.lexer.statemlexers.SettingSectionHeaderLexer` `lex_setting()` (`robot.parsing.lexer.context.TestCaseFileContext` `method`), 294 `method`), 289  
`lex()` (`robot.parsing.lexer.statemlexers.SingleType` `lex_setting()` (`robot.parsing.lexer.context.TestCaseFileContext` `method`), 294 `method`), 289  
`lex()` (`robot.parsing.lexer.statemlexers.StatementLexer` `lex_setting()` (`robot.parsing.lexer.context.TestCaseFileContext` `method`), 294 `method`), 289  
`lex()` (`robot.parsing.lexer.statemlexers.TestCaseSectionHeaderLexer` `lex_setting()` (`robot.parsing.lexer.context.TestCaseFileContext` `method`), 295 `method`), 286  
`lex()` (`robot.parsing.lexer.statemlexers.TestOrKeywordSettingLexer` `lex_setting()` (`robot.parsing.lexer.context.TestCaseFileContext` `method`), 296 `method`), 284  
`lex()` (`robot.parsing.lexer.statemlexers.TryHeaderLexer` `lex_setting()` (`robot.parsing.lexer.context.TestCaseFileContext` `method`), 297 `method`), 287  
`lex()` (`robot.parsing.lexer.statemlexers.TypeAndArgumentLexer` `lex_setting()` (`robot.parsing.lexer.context.TestCaseFileContext` `method`), 294 `method`), 287  
`lex()` (`robot.parsing.lexer.statemlexers.VariableLexer` `lex_setting()` (`robot.parsing.lexer.context.TestCaseFileContext` `method`), 296 `method`), 285  
`lex()` (`robot.parsing.lexer.statemlexers.VariableSectionHeaderLexer` `lex_setting()` (`robot.parsing.lexer.context.TestCaseFileContext` `method`), 295 `method`), 287  
`lex()` (`robot.parsing.lexer.statemlexers.WhileHeaderLexer` `lex_setting()` (`robot.parsing.lexer.context.TestCaseFileContext` `method`), 298 `method`), 286  
`lex_invalid()` (`robot.parsing.lexer.sections.InitFileSections` `lex_setting()` (`robot.parsing.lexer.context.TestCaseFileContext` `method`), 292 `method`), 285  
`lex_invalid()` (`robot.parsing.lexer.sections.ResourceFileSections` `lex_setting()` (`robot.parsing.lexer.context.TestCaseFileContext` `method`), 291 `method`), 287  
`lex_invalid()` (`robot.parsing.lexer.sections.Sections` `lex_setting()` (`robot.parsing.lexer.context.TestCaseFileContext` `method`), 291 `method`), 284  
`lex_invalid()` (`robot.parsing.lexer.sections.TestCaseFileSections` `lex_setting()` (`robot.parsing.lexer.context.TestCaseFileContext` `method`), 291 `method`), 284  
`lex_invalid_section()` (`robot.parsing.lexer.context.FileContext` `lex_setting()` (`robot.parsing.lexer.context.TestCaseFileContext` `method`), 288 `method`), 286  
`lex_invalid_section()` (`robot.parsing.lexer.context.InitFileContext` `lex_setting()` (`robot.parsing.lexer.context.TestCaseFileContext` `method`), 289 `method`), 285  
`lex_invalid_section()` (`robot.parsing.lexer.context.ResourceFileContext` `lex_setting()` (`robot.parsing.lexer.context.TestCaseFileContext` `method`), 289 `method`), 288  
`lex_invalid_section()` (`robot.parsing.lexer.context.TestCaseFileContext` `lex_setting()` (`robot.parsing.lexer.context.TestCaseFileContext` `method`), 288 `method`), 287

[lexer\\_for\(\) \(robot.parsing.lexer.blocklexers.BlockLexer method\)](#), 283  
[lexer\\_for\(\) \(robot.parsing.lexer.blocklexers.CommentSectionLexer method\)](#), 285  
[lexer\\_for\(\) \(robot.parsing.lexer.blocklexers.ErrorSectionLexer method\)](#), 286  
[lexer\\_for\(\) \(robot.parsing.lexer.blocklexers.FileLexer method\)](#), 284  
[lexer\\_for\(\) \(robot.parsing.lexer.blocklexers.ForLexer method\)](#), 287  
[lexer\\_for\(\) \(robot.parsing.lexer.blocklexers.IfLexer method\)](#), 287  
[lexer\\_for\(\) \(robot.parsing.lexer.blocklexers.ImplicitCommentSectionLexer method\)](#), 285  
[lexer\\_for\(\) \(robot.parsing.lexer.blocklexers.InlineIfLexer method\)](#), 288  
[lexer\\_for\(\) \(robot.parsing.lexer.blocklexers.KeywordLexer method\)](#), 286  
[lexer\\_for\(\) \(robot.parsing.lexer.blocklexers.KeywordSectionLexer method\)](#), 285  
[lexer\\_for\(\) \(robot.parsing.lexer.blocklexers.NestedBlockLexer method\)](#), 287  
[lexer\\_for\(\) \(robot.parsing.lexer.blocklexers.SectionLexer method\)](#), 284  
[lexer\\_for\(\) \(robot.parsing.lexer.blocklexers.SettingSectionLexer method\)](#), 284  
[lexer\\_for\(\) \(robot.parsing.lexer.blocklexers.TestCaseLexer method\)](#), 286  
[lexer\\_for\(\) \(robot.parsing.lexer.blocklexers.TestCaseSectionLexer method\)](#), 285  
[lexer\\_for\(\) \(robot.parsing.lexer.blocklexers.TestOrKeywordParser method\)](#), 286  
[lexer\\_for\(\) \(robot.parsing.lexer.blocklexers.TryLexer method\)](#), 288  
[lexer\\_for\(\) \(robot.parsing.lexer.blocklexers.VariableSectionLexer method\)](#), 284  
[lexer\\_for\(\) \(robot.parsing.lexer.blocklexers.WhileLexer method\)](#), 287  
[LexingContext \(class in robot.parsing.lexer.context\)](#), 288  
[LibDoc \(class in robot.libdoc\)](#), 580  
[libdoc\(\) \(in module robot.libdoc\)](#), 581  
[libdoc\\_cli\(\) \(in module robot.libdoc\)](#), 580  
[LibdocHtmlWriter \(class in robot.libdocpkg.htmlwriter\)](#), 37  
[LibdocJsonWriter \(class in robot.libdocpkg.jsonwriter\)](#), 37  
[LibdocModelWriter \(class in robot.libdocpkg.htmlwriter\)](#), 37  
[LibdocOutput \(class in robot.libdocpkg.output\)](#), 38  
[LibdocWriter\(\) \(in module robot.libdocpkg.writer\)](#), 38  
[LibdocXmlWriter \(class in robot.libdocpkg.xmlwriter\)](#), 39  
[libname \(robot.result.model.Break attribute\)](#), 453  
[libname \(robot.result.model.Continue attribute\)](#), 451  
[libname \(robot.result.model.For attribute\)](#), 433  
[libname \(robot.result.model.ForIteration attribute\)](#), 430  
[libname \(robot.result.model.If attribute\)](#), 442  
[libname \(robot.result.model.IfBranch attribute\)](#), 439  
[libname \(robot.result.model.Keyword attribute\)](#), 454  
[libname \(robot.result.model.Return attribute\)](#), 449  
[libname \(robot.result.model.Try attribute\)](#), 446  
[libname \(robot.result.model.TryBranch attribute\)](#), 444  
[libname \(robot.result.model.While attribute\)](#), 437  
[libname \(robot.result.model.WhileIteration attribute\)](#), 435  
[libname \(robot.result.model.deprecation.DeprecatedAttributesMixin attribute\)](#), 464  
[libname \(robot.running.librarykeywordrunner.EmbeddedArgumentsRunner attribute\)](#), 515  
[libname \(robot.running.librarykeywordrunner.LibraryKeywordRunner attribute\)](#), 514  
[libname \(robot.running.librarykeywordrunner.RunKeywordRunner attribute\)](#), 515  
[libname \(robot.running.userkeywordrunner.EmbeddedArgumentsRunner attribute\)](#), 552  
[libname \(robot.running.userkeywordrunner.UserKeywordRunner attribute\)](#), 552  
[libraries \(robot.running.namespace.Namespace attribute\)](#), 540  
[LIBRARY \(robot.parsing.lexer.tokens.END attribute\)](#), 304  
[LIBRARY \(robot.parsing.lexer.tokens.EOS attribute\)](#), 302  
[LIBRARY \(robot.parsing.lexer.tokens.Token attribute\)](#), 300  
[LIBRARY \(robot.running.handlers.EmbeddedArgumentsHandler attribute\)](#), 514  
[library \(robot.running.librarykeywordrunner.EmbeddedArgumentsRunner attribute\)](#), 515  
[library \(robot.running.librarykeywordrunner.LibraryKeywordRunner attribute\)](#), 514  
[library \(robot.running.librarykeywordrunner.RunKeywordRunner attribute\)](#), 515  
[library\(\) \(in module robot.api.deco\)](#), 11  
[library\(\) \(robot.running.model.Imports method\)](#), 539  
[LIBRARY\\_TYPE \(robot.running.handlerstore.HandlerStore attribute\)](#), 514  
[LibraryDoc \(class in robot.libdocpkg.model\)](#), 37  
[LibraryDocBuilder \(class in robot.libdocpkg.robotbuilder\)](#), 38  
[LibraryDocumentation\(\) \(in module robot.libdocpkg.builder\)](#), 35  
[LibraryImport \(class in robot.parsing.model.statements\)](#), 316  
[LibraryKeywordRunner \(class in robot.running.librarykeywordrunner\)](#), 514



- [robot.running.librarykeywordrunner](#)), 514
- [LibraryListenerMethods](#) (class in [robot.output.listenermethods](#)), 272
- [LibraryListeners](#) (class in [robot.output.listeners](#)), 272
- [LibraryScope\(\)](#) (in module [robot.running.libraryscopes](#)), 515
- [lift\(\)](#) ([robot.libraries.dialogs\\_py.InputDialog](#) method), 142
- [lift\(\)](#) ([robot.libraries.dialogs\\_py.MessageDialog](#) method), 128
- [lift\(\)](#) ([robot.libraries.dialogs\\_py.MultipleSelectionDialog](#) method), 170
- [lift\(\)](#) ([robot.libraries.dialogs\\_py.PassFailDialog](#) method), 184
- [lift\(\)](#) ([robot.libraries.dialogs\\_py.SelectionDialog](#) method), 156
- [limit](#) ([robot.model.control.While](#) attribute), 205
- [limit](#) ([robot.parsing.model.blocks.While](#) attribute), 310
- [limit](#) ([robot.parsing.model.statements.WhileHeader](#) attribute), 346
- [limit](#) ([robot.result.model.While](#) attribute), 437
- [limit](#) ([robot.running.model.While](#) attribute), 523
- [limit\\_exceeded\(\)](#) ([robot.running.bodyrunner.DurationLimit](#) method), 512
- [limit\\_exceeded\(\)](#) ([robot.running.bodyrunner.InvalidLimit](#) method), 512
- [limit\\_exceeded\(\)](#) ([robot.running.bodyrunner.IterationCountLimit](#) method), 512
- [limit\\_exceeded\(\)](#) ([robot.running.bodyrunner.NoLimit](#) method), 512
- [limit\\_exceeded\(\)](#) ([robot.running.bodyrunner.WhileLimit](#) method), 512
- [LineFormatter](#) (class in [robot.utils.htmlformatters](#)), 561
- [lineno](#) ([robot.model.testcase.TestCase](#) attribute), 248
- [lineno](#) ([robot.parsing.lexer.tokens.END](#) attribute), 305
- [lineno](#) ([robot.parsing.lexer.tokens.EOS](#) attribute), 303
- [lineno](#) ([robot.parsing.lexer.tokens.Token](#) attribute), 301
- [lineno](#) ([robot.parsing.model.blocks.Block](#) attribute), 306
- [lineno](#) ([robot.parsing.model.blocks.CommentSection](#) attribute), 308
- [lineno](#) ([robot.parsing.model.blocks.File](#) attribute), 306
- [lineno](#) ([robot.parsing.model.blocks.For](#) attribute), 309
- [lineno](#) ([robot.parsing.model.blocks.HeaderAndBody](#) attribute), 306
- [lineno](#) ([robot.parsing.model.blocks.If](#) attribute), 309
- [lineno](#) ([robot.parsing.model.blocks.Keyword](#) attribute), 308
- [lineno](#) ([robot.parsing.model.blocks.KeywordSection](#) attribute), 308
- [lineno](#) ([robot.parsing.model.blocks.Section](#) attribute), 307
- [lineno](#) ([robot.parsing.model.blocks.SettingSection](#) attribute), 307
- [lineno](#) ([robot.parsing.model.blocks.TestCase](#) attribute), 308
- [lineno](#) ([robot.parsing.model.blocks.TestCaseSection](#) attribute), 307
- [lineno](#) ([robot.parsing.model.blocks.Try](#) attribute), 310
- [lineno](#) ([robot.parsing.model.blocks.VariableSection](#) attribute), 307
- [lineno](#) ([robot.parsing.model.blocks.While](#) attribute), 310
- [lineno](#) ([robot.parsing.model.statements.Arguments](#) attribute), 334
- [lineno](#) ([robot.parsing.model.statements.Break](#) attribute), 350
- [lineno](#) ([robot.parsing.model.statements.Comment](#) attribute), 351
- [lineno](#) ([robot.parsing.model.statements.Continue](#) attribute), 349
- [lineno](#) ([robot.parsing.model.statements.DefaultTags](#) attribute), 322
- [lineno](#) ([robot.parsing.model.statements.Documentation](#) attribute), 319
- [lineno](#) ([robot.parsing.model.statements.DocumentationOrMetadata](#) attribute), 312
- [lineno](#) ([robot.parsing.model.statements.ElseHeader](#) attribute), 342
- [lineno](#) ([robot.parsing.model.statements.ElseIfHeader](#) attribute), 341
- [lineno](#) ([robot.parsing.model.statements.EmptyLine](#) attribute), 352
- [lineno](#) ([robot.parsing.model.statements.End](#) attribute), 346
- [lineno](#) ([robot.parsing.model.statements.Error](#) attribute), 352
- [lineno](#) ([robot.parsing.model.statements.ExceptHeader](#) attribute), 344
- [lineno](#) ([robot.parsing.model.statements.FinallyHeader](#) attribute), 345
- [lineno](#) ([robot.parsing.model.statements.Fixture](#) attribute), 315
- [lineno](#) ([robot.parsing.model.statements.ForceTags](#) attribute), 321
- [lineno](#) ([robot.parsing.model.statements.ForHeader](#) attribute), 337
- [lineno](#) ([robot.parsing.model.statements.IfElseHeader](#) attribute), 338
- [lineno](#) ([robot.parsing.model.statements.IfHeader](#) attribute), 339
- [lineno](#) ([robot.parsing.model.statements.InlineIfHeader](#) attribute), 340
- [lineno](#) ([robot.parsing.model.statements.KeywordCall](#) attribute), 336
- [lineno](#) ([robot.parsing.model.statements.KeywordName](#)

<a href="#">attribute</a> ), 329	<a href="#">attribute</a> ), 318
lineno ( <a href="#">robot.parsing.model.statements.LibraryImport attribute</a> ), 317	lineno ( <a href="#">robot.parsing.model.statements.WhileHeader attribute</a> ), 347
lineno ( <a href="#">robot.parsing.model.statements.LoopControl attribute</a> ), 348	lineno ( <a href="#">robot.result.model.TestCase attribute</a> ), 459
lineno ( <a href="#">robot.parsing.model.statements.Metadata attribute</a> ), 320	lineno ( <a href="#">robot.running.model.Break attribute</a> ), 531
lineno ( <a href="#">robot.parsing.model.statements.MultiValue attribute</a> ), 314	lineno ( <a href="#">robot.running.model.Continue attribute</a> ), 530
lineno ( <a href="#">robot.parsing.model.statements.NoArgumentHeader attribute</a> ), 342	lineno ( <a href="#">robot.running.model.For attribute</a> ), 520
lineno ( <a href="#">robot.parsing.model.statements.ResourceImport attribute</a> ), 317	lineno ( <a href="#">robot.running.model.If attribute</a> ), 524
lineno ( <a href="#">robot.parsing.model.statements.Return attribute</a> ), 335	lineno ( <a href="#">robot.running.model.IfBranch attribute</a> ), 523
lineno ( <a href="#">robot.parsing.model.statements.ReturnStatement attribute</a> ), 347	lineno ( <a href="#">robot.running.model.Keyword attribute</a> ), 518
lineno ( <a href="#">robot.parsing.model.statements.SectionHeader attribute</a> ), 316	lineno ( <a href="#">robot.running.model.Return attribute</a> ), 528
lineno ( <a href="#">robot.parsing.model.statements.Setup attribute</a> ), 330	lineno ( <a href="#">robot.running.model.TestCase attribute</a> ), 533
lineno ( <a href="#">robot.parsing.model.statements.SingleValue attribute</a> ), 313	lineno ( <a href="#">robot.running.model.Try attribute</a> ), 527
lineno ( <a href="#">robot.parsing.model.statements.Statement attribute</a> ), 311	lineno ( <a href="#">robot.running.model.TryBranch attribute</a> ), 526
lineno ( <a href="#">robot.parsing.model.statements.SuiteSetup attribute</a> ), 322	lineno ( <a href="#">robot.running.model.While attribute</a> ), 521
lineno ( <a href="#">robot.parsing.model.statements.SuiteTeardown attribute</a> ), 323	lines ( <a href="#">robot.parsing.model.statements.Arguments attribute</a> ), 334
lineno ( <a href="#">robot.parsing.model.statements.Tags attribute</a> ), 332	lines ( <a href="#">robot.parsing.model.statements.Break attribute</a> ), 350
lineno ( <a href="#">robot.parsing.model.statements.Teardown attribute</a> ), 331	lines ( <a href="#">robot.parsing.model.statements.Comment attribute</a> ), 351
lineno ( <a href="#">robot.parsing.model.statements.Template attribute</a> ), 332	lines ( <a href="#">robot.parsing.model.statements.Continue attribute</a> ), 349
lineno ( <a href="#">robot.parsing.model.statements.TemplateArgument attribute</a> ), 337	lines ( <a href="#">robot.parsing.model.statements.DefaultTags attribute</a> ), 322
lineno ( <a href="#">robot.parsing.model.statements.TestCaseName attribute</a> ), 328	lines ( <a href="#">robot.parsing.model.statements.Documentation attribute</a> ), 319
lineno ( <a href="#">robot.parsing.model.statements.TestSetup attribute</a> ), 324	lines ( <a href="#">robot.parsing.model.statements.DocumentationOrMetadata attribute</a> ), 312
lineno ( <a href="#">robot.parsing.model.statements.TestTeardown attribute</a> ), 325	lines ( <a href="#">robot.parsing.model.statements.ElseHeader attribute</a> ), 342
lineno ( <a href="#">robot.parsing.model.statements.TestTemplate attribute</a> ), 326	lines ( <a href="#">robot.parsing.model.statements.ElseIfHeader attribute</a> ), 341
lineno ( <a href="#">robot.parsing.model.statements.TestTimeout attribute</a> ), 327	lines ( <a href="#">robot.parsing.model.statements.EmptyLine attribute</a> ), 352
lineno ( <a href="#">robot.parsing.model.statements.Timeout attribute</a> ), 333	lines ( <a href="#">robot.parsing.model.statements.End attribute</a> ), 346
lineno ( <a href="#">robot.parsing.model.statements.TryHeader attribute</a> ), 343	lines ( <a href="#">robot.parsing.model.statements.Error attribute</a> ), 352
lineno ( <a href="#">robot.parsing.model.statements.Variable attribute</a> ), 328	lines ( <a href="#">robot.parsing.model.statements.ExceptHeader attribute</a> ), 344
lineno ( <a href="#">robot.parsing.model.statements.VariablesImport attribute</a> ), 317	lines ( <a href="#">robot.parsing.model.statements.FinallyHeader attribute</a> ), 345
	lines ( <a href="#">robot.parsing.model.statements.Fixture attribute</a> ), 315
	lines ( <a href="#">robot.parsing.model.statements.ForceTags attribute</a> ), 321
	lines ( <a href="#">robot.parsing.model.statements.ForHeader attribute</a> ), 337
	lines ( <a href="#">robot.parsing.model.statements.IfElseHeader attribute</a> ), 338
	lines ( <a href="#">robot.parsing.model.statements.IfHeader attribute</a> ), 339
	lines ( <a href="#">robot.parsing.model.statements.InlineIfHeader attribute</a> ), 340

- [attribute\), 340](#)
- lines [\(robot.parsing.model.statements.KeywordCall attribute\), 336](#)
- lines [\(robot.parsing.model.statements.KeywordName attribute\), 329](#)
- lines [\(robot.parsing.model.statements.LibraryImport attribute\), 317](#)
- lines [\(robot.parsing.model.statements.LoopControl attribute\), 348](#)
- lines [\(robot.parsing.model.statements.Metadata attribute\), 320](#)
- lines [\(robot.parsing.model.statements.MultiValue attribute\), 314](#)
- lines [\(robot.parsing.model.statements.NoArgumentHeader attribute\), 342](#)
- lines [\(robot.parsing.model.statements.ResourceImport attribute\), 317](#)
- lines [\(robot.parsing.model.statements.Return attribute\), 335](#)
- lines [\(robot.parsing.model.statements.ReturnStatement attribute\), 347](#)
- lines [\(robot.parsing.model.statements.SectionHeader attribute\), 316](#)
- lines [\(robot.parsing.model.statements.Setup attribute\), 330](#)
- lines [\(robot.parsing.model.statements.SingleValue attribute\), 313](#)
- lines [\(robot.parsing.model.statements.Statement attribute\), 312](#)
- lines [\(robot.parsing.model.statements.SuiteSetup attribute\), 322](#)
- lines [\(robot.parsing.model.statements.SuiteTeardown attribute\), 323](#)
- lines [\(robot.parsing.model.statements.Tags attribute\), 332](#)
- lines [\(robot.parsing.model.statements.Teardown attribute\), 331](#)
- lines [\(robot.parsing.model.statements.Template attribute\), 332](#)
- lines [\(robot.parsing.model.statements.TemplateArguments attribute\), 337](#)
- lines [\(robot.parsing.model.statements.TestCaseName attribute\), 328](#)
- lines [\(robot.parsing.model.statements.TestSetup attribute\), 324](#)
- lines [\(robot.parsing.model.statements.TestTeardown attribute\), 325](#)
- lines [\(robot.parsing.model.statements.TestTemplate attribute\), 326](#)
- lines [\(robot.parsing.model.statements.TestTimeout attribute\), 327](#)
- lines [\(robot.parsing.model.statements.Timeout attribute\), 333](#)
- lines [\(robot.parsing.model.statements.TryHeader attribute\), 343](#)
- lines [\(robot.parsing.model.statements.Variable attribute\), 328](#)
- lines [\(robot.parsing.model.statements.VariablesImport attribute\), 318](#)
- lines [\(robot.parsing.model.statements.WhileHeader attribute\), 347](#)
- [LineWriter \(class in robot.htmldata.htmlfilewriter\), 34](#)
- [link\(\) \(robot.reporting.jsbuildingcontext.JsBuildingContext method\), 357](#)
- [LinkFormatter \(class in robot.utils.htmlformatters\), 561](#)
- [links \(robot.model.stats.TagStat attribute\), 241](#)
- [list\(\) \(robot.libdocpkg.consoleviewer.ConsoleViewer method\), 36](#)
- [list\\_directories\\_in\\_directory\(\) \(robot.libraries.OperatingSystem.OperatingSystem method\), 87](#)
- [list\\_directory\(\) \(robot.libraries.OperatingSystem.OperatingSystem method\), 87](#)
- [list\\_files\\_in\\_directory\(\) \(robot.libraries.OperatingSystem.OperatingSystem method\), 87](#)
- [list\\_should\\_contain\\_sub\\_list\(\) \(robot.libraries.Collections.Collections method\), 70](#)
- [list\\_should\\_contain\\_value\(\) \(robot.libraries.Collections.Collections method\), 70](#)
- [list\\_should\\_not\\_contain\\_duplicates\(\) \(robot.libraries.Collections.Collections method\), 70](#)
- [list\\_should\\_not\\_contain\\_value\(\) \(robot.libraries.Collections.Collections method\), 70](#)
- [ListConverter \(class in robot.running.arguments.typeconverters\), 500](#)
- [listener\(\) \(robot.libraries.Telnet.TelnetConnection method\), 110](#)
- [ListenerArguments \(class in robot.output.listenerarguments\), 271](#)
- [ListenerMethod \(class in robot.output.listenermethods\), 272](#)
- [ListenerMethods \(class in robot.output.listenermethods\), 272](#)
- [ListenerProxy \(class in robot.output.listeners\), 272](#)
- [Listeners \(class in robot.output.listeners\), 272](#)
- [listeners \(robot.conf.settings.RobotSettings attribute\), 31](#)
- [ListFormatter \(class in robot.utils.htmlformatters\), 562](#)
- [lists\\_should\\_be\\_equal\(\)](#)

- (*robot.libraries.Collections.Collections* method), 70
  - ListVariableTableValue (class in *robot.variables.tablesetter*), 573
  - Location (class in *robot.libraries.XML*), 122
  - log (*robot.conf.settings.RebotSettings* attribute), 33
  - log (*robot.conf.settings.RobotSettings* attribute), 32
  - log () (*robot.libraries.BuiltIn.BuiltIn* method), 49
  - log\_config (*robot.conf.settings.RebotSettings* attribute), 32
  - log\_dictionary () (*robot.libraries.Collections.Collections* method), 71
  - log\_element () (*robot.libraries.XML.XML* method), 122
  - log\_environment\_variables ()
    - (*robot.libraries.OperatingSystem.OperatingSystem* method), 85
  - log\_file () (*robot.libraries.OperatingSystem.OperatingSystem* method), 80
  - log\_level (*robot.conf.settings.RebotSettings* attribute), 33
  - log\_level (*robot.conf.settings.RobotSettings* attribute), 32
  - log\_list () (*robot.libraries.Collections.Collections* method), 71
  - log\_many () (*robot.libraries.BuiltIn.BuiltIn* method), 49
  - log\_message () (*robot.output.listeners.LibraryListeners* method), 272
  - log\_message () (*robot.output.listeners.Listeners* method), 272
  - log\_message () (*robot.output.logger.Logger* method), 273
  - log\_message () (*robot.output.xmllogger.XmlLogger* method), 278
  - log\_message () (*robot.reporting.outputwriter.OutputWriter* method), 361
  - log\_output () (*robot.output.logger.Logger* method), 273
  - log\_to\_console () (*robot.libraries.BuiltIn.BuiltIn* method), 49
  - log\_variables () (*robot.libraries.BuiltIn.BuiltIn* method), 49
  - Logger (class in *robot.output.logger*), 273
  - LoggerProxy (class in *robot.output.logger*), 274
  - login () (*robot.libraries.Telnet.TelnetConnection* method), 108
  - LogWriter (class in *robot.reporting.logreportwriters*), 359
  - longname (*robot.model.testcase.TestCase* attribute), 249
  - longname (*robot.model.testsuite.TestSuite* attribute), 250
  - longname (*robot.result.model.TestCase* attribute), 459
  - longname (*robot.result.model.TestSuite* attribute), 461
  - longname (*robot.running.librarykeywordrunner.EmbeddedArgumentsRunner* attribute), 515
  - longname (*robot.running.librarykeywordrunner.LibraryKeywordRunner* attribute), 514
  - longname (*robot.running.librarykeywordrunner.RunKeywordRunner* attribute), 515
  - longname (*robot.running.model.TestCase* attribute), 533
  - longname (*robot.running.model.TestSuite* attribute), 537
  - longname (*robot.running.usererrorhandler.UserErrorHandler* attribute), 551
  - longname (*robot.running.userkeyword.EmbeddedArgumentsHandler* attribute), 552
  - longname (*robot.running.userkeyword.UserKeywordHandler* attribute), 552
  - longname (*robot.running.userkeywordrunner.EmbeddedArgumentsRunner* attribute), 552
  - longname (*robot.running.userkeywordrunner.UserKeywordRunner* attribute), 552
  - LoopControl (class in *robot.parsing.model.statements*), 348
  - lower () (*robot.libraries.dialogs\_py.InputDialog* method), 142
  - lower () (*robot.libraries.dialogs\_py.MessageDialog* method), 128
  - lower () (*robot.libraries.dialogs\_py.MultipleSelectionDialog* method), 170
  - lower () (*robot.libraries.dialogs\_py.PassFailDialog* method), 184
  - lower () (*robot.libraries.dialogs\_py.SelectionDialog* method), 156
- ## M
- main () (*robot.libdoc.LibDoc* method), 580
  - main () (*robot.rebot.Rebot* method), 582
  - main () (*robot.run.RobotFramework* method), 583
  - main () (*robot.testdoc.TestDoc* method), 585
  - main () (*robot.utils.application.Application* method), 553
  - mainloop () (*robot.libraries.dialogs\_py.InputDialog* method), 142
  - mainloop () (*robot.libraries.dialogs\_py.MessageDialog* method), 128
  - mainloop () (*robot.libraries.dialogs\_py.MultipleSelectionDialog* method), 170
  - mainloop () (*robot.libraries.dialogs\_py.PassFailDialog* method), 184
  - mainloop () (*robot.libraries.dialogs\_py.SelectionDialog* method), 156
  - make\_connection ()
    - (*robot.libraries.Remote.TimeoutHTTPSTransport* method), 95



[make\\_connection\(\)](#) ([robot.libraries.Remote.TimeoutHTTPTransport](#) method), 94  
[manage\(\)](#) ([robot.libraries.dialogs\\_py.InputDialog](#) method), 142  
[manage\(\)](#) ([robot.libraries.dialogs\\_py.MessageDialog](#) method), 128  
[manage\(\)](#) ([robot.libraries.dialogs\\_py.MultipleSelectionDialog](#) method), 170  
[manage\(\)](#) ([robot.libraries.dialogs\\_py.PassFailDialog](#) method), 184  
[manage\(\)](#) ([robot.libraries.dialogs\\_py.SelectionDialog](#) method), 156  
[map\(\)](#) ([robot.running.arguments.argumentmapper.ArgumentMapper](#) method), 492  
[map\(\)](#) ([robot.running.arguments.argumentspec.Argumentspec](#) method), 494  
[MappingDumper](#) (class in [robot.htmldata.jsonwriter](#)), 35  
[mark\(\)](#) ([robot.output.console.verbose.KeywordMarker](#) method), 269  
[match](#) ([robot.variables.search.VariableMatch](#) attribute), 572  
[match\(\)](#) ([robot.model.namepatterns.NamePatterns](#) method), 235  
[match\(\)](#) ([robot.model.namepatterns.SuiteNamePatterns](#) method), 235  
[match\(\)](#) ([robot.model.namepatterns.TestNamePatterns](#) method), 235  
[match\(\)](#) ([robot.model.stats.CombinedTagStat](#) method), 241  
[match\(\)](#) ([robot.model.tags.AndTagPattern](#) method), 242  
[match\(\)](#) ([robot.model.tags.NotTagPattern](#) method), 242  
[match\(\)](#) ([robot.model.tags.OrTagPattern](#) method), 242  
[match\(\)](#) ([robot.model.tags.SingleTagPattern](#) method), 242  
[match\(\)](#) ([robot.model.tags.TagPatterns](#) method), 242  
[match\(\)](#) ([robot.model.tags.Tags](#) method), 242  
[match\(\)](#) ([robot.model.tagstatistics.TagStatDoc](#) method), 247  
[match\(\)](#) ([robot.model.tagstatistics.TagStatLink](#) method), 247  
[match\(\)](#) ([robot.reporting.expandkeywordmatcher.ExpandKeywordMatcher](#) method), 357  
[match\(\)](#) ([robot.result.flattenkeywordmatcher.FlattenByName](#) method), 379  
[match\(\)](#) ([robot.result.flattenkeywordmatcher.FlattenByTag](#) method), 379  
[match\(\)](#) ([robot.result.flattenkeywordmatcher.FlattenByType](#) method), 379  
[match\(\)](#) ([robot.utils.htmlformatters.HeaderFormatter](#) method), 561  
[match\(\)](#) ([robot.utils.htmlformatters.RulerFormatter](#) method), 561  
[match\(\)](#) ([robot.utils.match.Matcher](#) method), 565  
[match\(\)](#) ([robot.utils.match.MultiMatcher](#) method), 565  
[match\\_any\(\)](#) ([robot.utils.match.Matcher](#) method), 565  
[match\\_any\(\)](#) ([robot.utils.match.MultiMatcher](#) method), 565  
[Matcher](#) (class in [robot.utils.match](#)), 565  
[matches\(\)](#) ([robot.running.handlers.EmbeddedArgumentsHandler](#) method), 514  
[matches\(\)](#) ([robot.running.userkeyword.EmbeddedArgumentsHandler](#) method), 552  
[max\\_assign\\_length](#) ([robot.conf.settings.RobotSettings](#) attribute), 32  
[max\\_error\\_lines](#) ([robot.conf.settings.RobotSettings](#) attribute), 32  
[maxargs](#) ([robot.running.arguments.argumentspec.Argumentspec](#) attribute), 494  
[maxsize\(\)](#) ([robot.libraries.dialogs\\_py.InputDialog](#) method), 142  
[maxsize\(\)](#) ([robot.libraries.dialogs\\_py.MessageDialog](#) method), 128  
[maxsize\(\)](#) ([robot.libraries.dialogs\\_py.MultipleSelectionDialog](#) method), 170  
[maxsize\(\)](#) ([robot.libraries.dialogs\\_py.PassFailDialog](#) method), 184  
[maxsize\(\)](#) ([robot.libraries.dialogs\\_py.SelectionDialog](#) method), 156  
[merge](#) ([robot.conf.settings.RebotSettings](#) attribute), 33  
[merge\(\)](#) ([robot.result.merger.Merger](#) method), 413  
[Merger](#) (class in [robot.result.merger](#)), 413  
[Message](#) (class in [robot.model.message](#)), 227  
[Message](#) (class in [robot.output.loggerhelper](#)), 274  
[Message](#) (class in [robot.result.model](#)), 427  
[message](#) ([robot.errors.BreakLoop](#) attribute), 579  
[message](#) ([robot.errors.ContinueLoop](#) attribute), 579  
[message](#) ([robot.errors.DataError](#) attribute), 575  
[message](#) ([robot.errors.ExecutionFailed](#) attribute), 576  
[message](#) ([robot.errors.ExecutionFailures](#) attribute), 577  
[message](#) ([robot.errors.ExecutionPassed](#) attribute), 578  
[message](#) ([robot.errors.ExecutionStatus](#) attribute), 576  
[message](#) ([robot.errors.FrameworkError](#) attribute), 575  
[message](#) ([robot.errors.HandlerExecutionFailed](#) attribute), 577  
[message](#) ([robot.errors.Information](#) attribute), 576  
[message](#) ([robot.errors.KeywordError](#) attribute), 575  
[message](#) ([robot.errors.PassExecution](#) attribute), 578  
[message](#) ([robot.errors.RemoteError](#) attribute), 580  
[message](#) ([robot.errors.ReturnFromKeyword](#) attribute), 579  
[message](#) ([robot.errors.RobotError](#) attribute), 574  
[message](#) ([robot.errors.TimeoutError](#) attribute), 575  
[message](#) ([robot.errors.UserKeywordExecutionFailed](#) attribute), 577  
[message](#) ([robot.errors.VariableError](#) attribute), 575

- MESSAGE (*robot.model.body.BodyItem* attribute), 193
- MESSAGE (*robot.model.control.Break* attribute), 214
- MESSAGE (*robot.model.control.Continue* attribute), 213
- MESSAGE (*robot.model.control.For* attribute), 204
- MESSAGE (*robot.model.control.If* attribute), 208
- MESSAGE (*robot.model.control.IfBranch* attribute), 207
- MESSAGE (*robot.model.control.Return* attribute), 212
- MESSAGE (*robot.model.control.Try* attribute), 211
- MESSAGE (*robot.model.control.TryBranch* attribute), 209
- MESSAGE (*robot.model.control.While* attribute), 205
- MESSAGE (*robot.model.keyword.Keyword* attribute), 225
- MESSAGE (*robot.model.message.Message* attribute), 228
- message (*robot.model.message.Message* attribute), 227
- message (*robot.model.totalstatistics.TotalStatistics* attribute), 253
- MESSAGE (*robot.output.loggerhelper.Message* attribute), 275
- message (*robot.output.loggerhelper.Message* attribute), 274
- MESSAGE (*robot.result.model.Break* attribute), 452
- message (*robot.result.model.Break* attribute), 453
- MESSAGE (*robot.result.model.Continue* attribute), 450
- message (*robot.result.model.Continue* attribute), 451
- MESSAGE (*robot.result.model.For* attribute), 431
- message (*robot.result.model.For* attribute), 433
- MESSAGE (*robot.result.model.ForIteration* attribute), 429
- message (*robot.result.model.ForIteration* attribute), 430
- MESSAGE (*robot.result.model.If* attribute), 441
- message (*robot.result.model.If* attribute), 442
- MESSAGE (*robot.result.model.IfBranch* attribute), 438
- message (*robot.result.model.IfBranch* attribute), 439
- MESSAGE (*robot.result.model.Keyword* attribute), 455
- message (*robot.result.model.Keyword* attribute), 454
- MESSAGE (*robot.result.model.Message* attribute), 427
- message (*robot.result.model.Message* attribute), 428
- MESSAGE (*robot.result.model.Return* attribute), 447
- message (*robot.result.model.Return* attribute), 449
- message (*robot.result.model.TestCase* attribute), 457
- message (*robot.result.model.TestSuite* attribute), 460
- MESSAGE (*robot.result.model.Try* attribute), 445
- message (*robot.result.model.Try* attribute), 446
- MESSAGE (*robot.result.model.TryBranch* attribute), 443
- message (*robot.result.model.TryBranch* attribute), 444
- MESSAGE (*robot.result.model.While* attribute), 436
- message (*robot.result.model.While* attribute), 437
- MESSAGE (*robot.result.model.WhileIteration* attribute), 434
- message (*robot.result.model.WhileIteration* attribute), 435
- message (*robot.result.model.deprecation.DeprecatedAttribute* attribute), 464
- MESSAGE (*robot.running.model.Break* attribute), 531
- MESSAGE (*robot.running.model.Continue* attribute), 530
- MESSAGE (*robot.running.model.For* attribute), 520
- MESSAGE (*robot.running.model.If* attribute), 525
- MESSAGE (*robot.running.model.IfBranch* attribute), 523
- MESSAGE (*robot.running.model.Keyword* attribute), 518
- MESSAGE (*robot.running.model.Return* attribute), 529
- MESSAGE (*robot.running.model.Try* attribute), 527
- MESSAGE (*robot.running.model.TryBranch* attribute), 526
- MESSAGE (*robot.running.model.While* attribute), 522
- message (*robot.running.status.ParentMessage* attribute), 547
- message (*robot.running.status.SuiteMessage* attribute), 546
- message (*robot.running.status.SuiteStatus* attribute), 545
- message (*robot.running.status.TestMessage* attribute), 546
- message (*robot.running.status.TestStatus* attribute), 546
- message (*robot.utils.error.ErrorDetails* attribute), 560
- message () (*robot.output.console.dotted.DottedOutput* method), 264
- message () (*robot.output.console.quiet.QuietOutput* method), 269
- message () (*robot.output.console.verbose.VerboseOutput* method), 269
- message () (*robot.output.console.verbose.VerboseWriter* method), 269
- message () (*robot.output.filelogger.FileLogger* method), 270
- message () (*robot.output.logger.Logger* method), 273
- message () (*robot.output.loggerhelper.AbstractLogger* method), 274
- message () (*robot.output.output.Output* method), 276
- message () (*robot.output.xmllogger.XmlLogger* method), 278
- message () (*robot.reporting.outputwriter.OutputWriter* method), 361
- message\_class (*robot.model.body.BaseBody* attribute), 194
- message\_class (*robot.model.body.Body* attribute), 197
- message\_class (*robot.model.body.Branches* attribute), 199
- message\_class (*robot.result.model.Body* attribute), 423
- message\_class (*robot.result.model.Branches* attribute), 425
- message\_class (*robot.result.model.Iterations* attribute), 426
- message\_class (*robot.running.model.Body* attribute), 517

`message_level()` (*robot.reporting.jsbuildingcontext.JsBuildingContext* attribute), 357  
`MessageArguments` (class in *robot.output.listenerarguments*), 271  
`MessageBuilder` (class in *robot.reporting.jsmodelbuilders*), 358  
`MessageDialog` (class in *robot.libraries.dialogs\_py*), 123  
`MessageFilter` (class in *robot.result.messagefilter*), 417  
`MessageHandler` (class in *robot.result.xmlelementhandlers*), 487  
`Messages` (class in *robot.model.message*), 229  
`messages` (*robot.result.executionerrors.ExecutionErrors* attribute), 376  
`messages` (*robot.result.model.Keyword* attribute), 454  
`Metadata` (class in *robot.model.metadata*), 229  
`Metadata` (class in *robot.parsing.model.statements*), 319  
`metadata` (*robot.model.testsuite.TestSuite* attribute), 250  
`METADATA` (*robot.parsing.lexer.tokens.END* attribute), 304  
`METADATA` (*robot.parsing.lexer.tokens.EOS* attribute), 302  
`METADATA` (*robot.parsing.lexer.tokens.Token* attribute), 299  
`metadata` (*robot.result.model.TestSuite* attribute), 461  
`metadata` (*robot.running.model.TestSuite* attribute), 537  
`MetadataHandler` (class in *robot.result.xmlelementhandlers*), 487  
`MetadataItemHandler` (class in *robot.result.xmlelementhandlers*), 487  
`MetaHandler` (class in *robot.result.xmlelementhandlers*), 488  
`minargs` (*robot.running.arguments.argumentspec.Argumentspec* attribute), 494  
`minsize()` (*robot.libraries.dialogs\_py.InputDialog* method), 142  
`minsize()` (*robot.libraries.dialogs\_py.MessageDialog* method), 129  
`minsize()` (*robot.libraries.dialogs\_py.MultipleSelectionDialog* method), 170  
`minsize()` (*robot.libraries.dialogs\_py.PassFailDialog* method), 184  
`minsize()` (*robot.libraries.dialogs\_py.SelectionDialog* method), 156  
`model_class` (*robot.parsing.parser.fileparser.CommentSectionParser* attribute), 355  
`model_class` (*robot.parsing.parser.fileparser.KeywordSectionParser* attribute), 355  
`model_class` (*robot.parsing.parser.fileparser.SectionParser* attribute), 355  
`ModelBuilder` (class in *robot.parsing.parser.fileparser.SettingSectionParser* attribute), 355  
`model_class` (*robot.parsing.parser.fileparser.TestCaseSectionParser* attribute), 355  
`model_class` (*robot.parsing.parser.fileparser.VariableSectionParser* attribute), 355  
`model_class()` (*robot.parsing.parser.fileparser.ImplicitCommentSectionParser* method), 355  
`ModelCombiner` (class in *robot.running.modelcombiner*), 539  
`ModelModifier` (class in *robot.model.modifier*), 230  
`ModelObject` (class in *robot.model.modelobject*), 230  
`ModelTransformer` (class in *robot.parsing.model.visitor*), 353  
`ModelValidator` (class in *robot.parsing.model.blocks*), 310  
`ModelVisitor` (class in *robot.parsing.model.visitor*), 353  
`ModelWriter` (class in *robot.htmldata.htmlfilewriter*), 33  
`ModelWriter` (class in *robot.parsing.model.blocks*), 310  
`move_directory()` (*robot.libraries.OperatingSystem.OperatingSystem* method), 84  
`move_file()` (*robot.libraries.OperatingSystem.OperatingSystem* method), 83  
`move_files()` (*robot.libraries.OperatingSystem.OperatingSystem* method), 84  
`move_to_end()` (*robot.utils.dotdict.DotDict* method), 559  
`msg()` (*robot.libraries.Telnet.TelnetConnection* method), 110  
`mt_interact()` (*robot.libraries.Telnet.TelnetConnection* method), 110  
`multi_use` (*robot.parsing.lexer.settings.InitFileSettings* attribute), 292  
`multi_use` (*robot.parsing.lexer.settings.KeywordSettings* attribute), 293  
`multi_use` (*robot.parsing.lexer.settings.ResourceFileSettings* attribute), 293  
`multi_use` (*robot.parsing.lexer.settings.Settings* attribute), 292  
`multi_use` (*robot.parsing.lexer.settings.TestCaseFileSettings* attribute), 292  
`multi_use` (*robot.parsing.lexer.settings.TestCaseSettings* attribute), 293  
`MultiMatcher` (class in *robot.utils.match*), 565  
`MultipleSelectionDialog` (class in *robot.libraries.dialogs\_py*), 164  
`MultiValue` (class in *robot.parsing.model.statements*), 313  
`name` (*robot.model.keyword.Keyword* attribute), 224

`name (robot.model.stats.Stat attribute)`, 239  
`name (robot.model.testcase.TestCase attribute)`, 247  
`name (robot.model.testsuite.TestSuite attribute)`, 250  
`name (robot.output.pyloggingconf.RobotHandler attribute)`, 278  
`NAME (robot.parsing.lexer.tokens.END attribute)`, 304  
`NAME (robot.parsing.lexer.tokens.EOS attribute)`, 302  
`NAME (robot.parsing.lexer.tokens.Token attribute)`, 300  
`name (robot.parsing.model.blocks.Keyword attribute)`, 308  
`name (robot.parsing.model.blocks.TestCase attribute)`, 308  
`name (robot.parsing.model.statements.Fixture attribute)`, 314  
`name (robot.parsing.model.statements.KeywordName attribute)`, 329  
`name (robot.parsing.model.statements.LibraryImport attribute)`, 316  
`name (robot.parsing.model.statements.Metadata attribute)`, 319  
`name (robot.parsing.model.statements.ResourceImport attribute)`, 317  
`name (robot.parsing.model.statements.SectionHeader attribute)`, 315  
`name (robot.parsing.model.statements.Setup attribute)`, 330  
`name (robot.parsing.model.statements.SuiteSetup attribute)`, 322  
`name (robot.parsing.model.statements.SuiteTeardown attribute)`, 323  
`name (robot.parsing.model.statements.Teardown attribute)`, 331  
`name (robot.parsing.model.statements.TestCaseName attribute)`, 328  
`name (robot.parsing.model.statements.TestSetup attribute)`, 324  
`name (robot.parsing.model.statements.TestTeardown attribute)`, 325  
`name (robot.parsing.model.statements.Variable attribute)`, 327  
`name (robot.parsing.model.statements.VariablesImport attribute)`, 318  
`name (robot.result.model.Break attribute)`, 453  
`name (robot.result.model.Continue attribute)`, 451  
`name (robot.result.model.For attribute)`, 431  
`name (robot.result.model.ForIteration attribute)`, 429  
`name (robot.result.model.If attribute)`, 442  
`name (robot.result.model.IfBranch attribute)`, 438  
`name (robot.result.model.Keyword attribute)`, 455  
`name (robot.result.model.Return attribute)`, 449  
`name (robot.result.model.TestCase attribute)`, 459  
`name (robot.result.model.TestSuite attribute)`, 461  
`name (robot.result.model.Try attribute)`, 446  
`name (robot.result.model.TryBranch attribute)`, 442  
`name (robot.result.model.While attribute)`, 436  
`name (robot.result.model.WhileIteration attribute)`, 433  
`name (robot.result.model.deprecation.DeprecatedAttributesMixin attribute)`, 464  
`name (robot.running.arguments.customconverters.ConverterInfo attribute)`, 495  
`name (robot.running.dynamicmethods.GetKeywordArguments attribute)`, 513  
`name (robot.running.dynamicmethods.GetKeywordDocumentation attribute)`, 513  
`name (robot.running.dynamicmethods.GetKeywordNames attribute)`, 513  
`name (robot.running.dynamicmethods.GetKeywordSource attribute)`, 513  
`name (robot.running.dynamicmethods.GetKeywordTags attribute)`, 513  
`name (robot.running.dynamicmethods.GetKeywordTypes attribute)`, 513  
`name (robot.running.dynamicmethods.RunKeyword attribute)`, 513  
`name (robot.running.model.Keyword attribute)`, 519  
`name (robot.running.model.TestCase attribute)`, 533  
`name (robot.running.model.TestSuite attribute)`, 537  
`name (robot.variables.search.VariableMatch attribute)`, 572  
`name_and_arguments (robot.parsing.lexer.settings.InitFileSettings attribute)`, 292  
`name_and_arguments (robot.parsing.lexer.settings.KeywordSettings attribute)`, 293  
`name_and_arguments (robot.parsing.lexer.settings.ResourceFileSettings attribute)`, 293  
`name_and_arguments (robot.parsing.lexer.settings.Settings attribute)`, 292  
`name_and_arguments (robot.parsing.lexer.settings.TestCaseFileSettings attribute)`, 292  
`name_and_arguments (robot.parsing.lexer.settings.TestCaseSettings attribute)`, 293  
`name_arguments_and_with_name (robot.parsing.lexer.settings.InitFileSettings attribute)`, 292  
`name_arguments_and_with_name (robot.parsing.lexer.settings.KeywordSettings attribute)`, 293  
`name_arguments_and_with_name (robot.parsing.lexer.settings.ResourceFileSettings attribute)`, 293  
`name_arguments_and_with_name (robot.parsing.lexer.settings.Settings attribute)`,



292

name\_arguments\_and\_with\_name (robot.parsing.lexer.settings.TestCaseFileSettings attribute), 292

name\_arguments\_and\_with\_name (robot.parsing.lexer.settings.TestCaseSettings attribute), 293

name\_type (robot.parsing.lexer.blocklexers.KeywordLexer attribute), 286

name\_type (robot.parsing.lexer.blocklexers.TestCaseLexer attribute), 286

name\_type (robot.parsing.lexer.blocklexers.TestOrKeywordLexer attribute), 286

NAMED\_ONLY (robot.running.arguments.argumentspec.ArgInfo attribute), 494

NAMED\_ONLY\_MARKER (robot.running.arguments.argumentspec.ArgInfo attribute), 494

NamedArgumentResolver (class in robot.running.arguments.argumentresolver), 493

NamePatterns (class in robot.model.namepatterns), 235

names (robot.parsing.lexer.settings.InitFileSettings attribute), 292

names (robot.parsing.lexer.settings.KeywordSettings attribute), 293

names (robot.parsing.lexer.settings.ResourceFileSettings attribute), 293

names (robot.parsing.lexer.settings.Settings attribute), 292

names (robot.parsing.lexer.settings.TestCaseFileSettings attribute), 292

names (robot.parsing.lexer.settings.TestCaseSettings attribute), 293

Namespace (class in robot.running.namespace), 540

namespaces (robot.running.context.ExecutionContext attribute), 513

NameSpaceStripper (class in robot.libraries.XML), 122

nametowidget () (robot.libraries.dialogs\_py.InputDialog method), 142

nametowidget () (robot.libraries.dialogs\_py.MessageDialog method), 129

nametowidget () (robot.libraries.dialogs\_py.MultipleSelectionDialog method), 170

nametowidget () (robot.libraries.dialogs\_py.PassFailDialog method), 184

nametowidget () (robot.libraries.dialogs\_py.SelectionDialog method), 156

NestedBlockLexer (class in robot.parsing.lexer.blocklexers), 286

NestedBlockParser (class in robot.parsing.parser.blockparsers), 354

NEW\_ENVIRON\_IS (robot.libraries.Telnet.TelnetConnection attribute), 107

NEW\_ENVIRON\_VALUE (robot.libraries.Telnet.TelnetConnection attribute), 107

NEW\_ENVIRON\_VAR (robot.libraries.Telnet.TelnetConnection attribute), 107

new\_suite\_scope () (robot.output.listenermethods.LibraryListenerMethods method), 272

new\_suite\_scope () (robot.output.listeners.LibraryListeners method), 272

newline (robot.utils.htmlformatters.LineFormatter attribute), 561

no\_conversion\_needed () (robot.running.arguments.typeconverters.BooleanConverter method), 497

no\_conversion\_needed () (robot.running.arguments.typeconverters.ByteArrayConverter method), 499

no\_conversion\_needed () (robot.running.arguments.typeconverters.BytesConverter method), 498

no\_conversion\_needed () (robot.running.arguments.typeconverters.CombinedConverter method), 502

no\_conversion\_needed () (robot.running.arguments.typeconverters.CustomConverter method), 503

no\_conversion\_needed () (robot.running.arguments.typeconverters.DateConverter method), 499

no\_conversion\_needed () (robot.running.arguments.typeconverters.DateTimeConverter method), 499

no\_conversion\_needed () (robot.running.arguments.typeconverters.DecimalConverter method), 498

no\_conversion\_needed () (robot.running.arguments.typeconverters.DictionaryConverter method), 501

no\_conversion\_needed () (robot.running.arguments.typeconverters.EnumConverter method), 496

no\_conversion\_needed () (robot.running.arguments.typeconverters.FloatConverter method), 497

no\_conversion\_needed () (robot.running.arguments.typeconverters.FrozenSetConverter method), 502

no\_conversion\_needed () (robot.running.arguments.typeconverters.IntegerConverter method), 497

`no_conversion_needed()` (*robot.running.arguments.typeconverters.ListConverter* method), 500

`no_conversion_needed()` (*robot.running.arguments.typeconverters.NoneConverter* method), 500

`no_conversion_needed()` (*robot.running.arguments.typeconverters.SetConverter* method), 502

`no_conversion_needed()` (*robot.running.arguments.typeconverters.StringConverter* method), 496

`no_conversion_needed()` (*robot.running.arguments.typeconverters.TimeDeltaConverter* method), 500

`no_conversion_needed()` (*robot.running.arguments.typeconverters.TupleConverter* method), 501

`no_conversion_needed()` (*robot.running.arguments.typeconverters.TypeConverter* method), 495

`no_dynamic_method()` (in module *robot.running.dynamichelpers*), 513

`no_operation()` (*robot.libraries.BuiltIn.BuiltIn* method), 50

`NoArgumentHeader` (class in *robot.parsing.model.statements*), 342

`NoConnection` (class in *robot.utils.connectioncache*), 558

`NoHighlighting` (class in *robot.output.console.highlighting*), 268

`NoInitFileDirectoryParser` (class in *robot.running.builder.parsers*), 505

`NoLimit` (class in *robot.running.bodyrunner*), 512

`NoLogger` (class in *robot.utils.importer*), 564

`NoMatchError`, 111

`non_ascii` (*robot.libraries.Remote.ArgumentCoercer* attribute), 94

`NON_DATA_TOKENS` (*robot.parsing.lexer.tokens.END* attribute), 304

`NON_DATA_TOKENS` (*robot.parsing.lexer.tokens.EOS* attribute), 302

`NON_DATA_TOKENS` (*robot.parsing.lexer.tokens.Token* attribute), 301

`NonDottedImporter` (class in *robot.utils.importer*), 563

`none_shall_pass()` (in module *robot.libraries.Easter*), 77

`NoneConverter` (class in *robot.running.arguments.typeconverters*), 500

`NoneDumper` (class in *robot.htmldata.jsonwriter*), 35

`NoOutput` (class in *robot.output.console.quiet*), 269

`NoReturnValueResolver` (class in *robot.variables.assigner*), 568

`normalize` (*robot.model.keyword.Keywords* attribute), 226

`normalize()` (in module *robot.utils.normalizing*), 566

`normalize_path()` (*robot.libraries.OperatingSystem.OperatingSystem* method), 85

`normalize_whitespace()` (in module *robot.utils.normalizing*), 566

`NormalizedDict` (class in *robot.utils.normalizing*), 566

`not_keyword()` (in module *robot.api.deco*), 10

`NOT_RUN` (*robot.result.model.Break* attribute), 452

`not_run` (*robot.result.model.Break* attribute), 453

`NOT_RUN` (*robot.result.model.Continue* attribute), 450

`not_run` (*robot.result.model.Continue* attribute), 451

`NOT_RUN` (*robot.result.model.For* attribute), 431

`not_run` (*robot.result.model.For* attribute), 433

`NOT_RUN` (*robot.result.model.ForIteration* attribute), 429

`not_run` (*robot.result.model.ForIteration* attribute), 430

`NOT_RUN` (*robot.result.model.If* attribute), 441

`not_run` (*robot.result.model.If* attribute), 442

`NOT_RUN` (*robot.result.model.IfBranch* attribute), 438

`not_run` (*robot.result.model.IfBranch* attribute), 440

`NOT_RUN` (*robot.result.model.Keyword* attribute), 455

`not_run` (*robot.result.model.Keyword* attribute), 456

`NOT_RUN` (*robot.result.model.Return* attribute), 448

`not_run` (*robot.result.model.Return* attribute), 449

`NOT_RUN` (*robot.result.model.StatusMixin* attribute), 428

`not_run` (*robot.result.model.StatusMixin* attribute), 428

`NOT_RUN` (*robot.result.model.TestCase* attribute), 457

`not_run` (*robot.result.model.TestCase* attribute), 457

`NOT_RUN` (*robot.result.model.TestSuite* attribute), 460

`not_run` (*robot.result.model.TestSuite* attribute), 461

`NOT_RUN` (*robot.result.model.Try* attribute), 445

`not_run` (*robot.result.model.Try* attribute), 446

`NOT_RUN` (*robot.result.model.TryBranch* attribute), 443

`not_run` (*robot.result.model.TryBranch* attribute), 444

`NOT_RUN` (*robot.result.model.While* attribute), 436

`not_run` (*robot.result.model.While* attribute), 437

`NOT_RUN` (*robot.result.model.WhileIteration* attribute), 434

`not_run` (*robot.result.model.WhileIteration* attribute), 435

`NOT_SET` (*robot.result.model.Break* attribute), 452

`NOT_SET` (*robot.result.model.Continue* attribute), 450

`NOT_SET` (*robot.result.model.For* attribute), 431

`NOT_SET` (*robot.result.model.ForIteration* attribute), 429

`NOT_SET` (*robot.result.model.If* attribute), 441

`NOT_SET` (*robot.result.model.IfBranch* attribute), 438

`NOT_SET` (*robot.result.model.Keyword* attribute), 455

`NOT_SET` (*robot.result.model.Return* attribute), 448

`NOT_SET` (*robot.result.model.StatusMixin* attribute), 428

- NOT\_SET (*robot.result.model.TestCase* attribute), 457
- NOT\_SET (*robot.result.model.TestSuite* attribute), 460
- NOT\_SET (*robot.result.model.Try* attribute), 445
- NOT\_SET (*robot.result.model.TryBranch* attribute), 443
- NOT\_SET (*robot.result.model.While* attribute), 436
- NOT\_SET (*robot.result.model.WhileIteration* attribute), 434
- NotSet (class in *robot.libraries.Collections*), 65
- NOTSET (*robot.running.arguments.argumentspec.ArgInfo* attribute), 494
- NotTagPattern (class in *robot.model.tags*), 242
- NullMarkupWriter (class in *robot.utils.markupwriters*), 565
- NullNamedArgumentResolver (class in *robot.running.arguments.argumentresolver*), 493
- NumberFinder (class in *robot.variables.finders*), 569
- numerator (*robot.reporting.stringcache.StringIndex* attribute), 365
- ## O
- OneReturnValueResolver (class in *robot.variables.assigner*), 568
- open () (*robot.libraries.Telnet.TelnetConnection* method), 110
- open\_connection () (*robot.libraries.Telnet.Telnet* method), 106
- OperatingSystem (class in *robot.libraries.OperatingSystem*), 78
- OPTION (*robot.parsing.lexer.tokens.END* attribute), 304
- OPTION (*robot.parsing.lexer.tokens.EOS* attribute), 302
- OPTION (*robot.parsing.lexer.tokens.Token* attribute), 301
- option\_add () (*robot.libraries.dialogs\_py.InputDialog* method), 143
- option\_add () (*robot.libraries.dialogs\_py.MessageDialog* method), 129
- option\_add () (*robot.libraries.dialogs\_py.MultipleSelectionDialog* method), 171
- option\_add () (*robot.libraries.dialogs\_py.PassFailDialog* method), 185
- option\_add () (*robot.libraries.dialogs\_py.SelectionDialog* method), 157
- option\_clear () (*robot.libraries.dialogs\_py.InputDialog* method), 143
- option\_clear () (*robot.libraries.dialogs\_py.MessageDialog* method), 129
- option\_clear () (*robot.libraries.dialogs\_py.MultipleSelectionDialog* method), 171
- option\_clear () (*robot.libraries.dialogs\_py.PassFailDialog* method), 185
- option\_clear () (*robot.libraries.dialogs\_py.SelectionDialog* method), 157
- option\_get () (*robot.libraries.dialogs\_py.InputDialog* method), 143
- option\_get () (*robot.libraries.dialogs\_py.MessageDialog* method), 129
- option\_get () (*robot.libraries.dialogs\_py.MultipleSelectionDialog* method), 171
- option\_get () (*robot.libraries.dialogs\_py.PassFailDialog* method), 185
- option\_get () (*robot.libraries.dialogs\_py.SelectionDialog* method), 157
- option\_readfile () (*robot.libraries.dialogs\_py.InputDialog* method), 143
- option\_readfile () (*robot.libraries.dialogs\_py.MessageDialog* method), 129
- option\_readfile () (*robot.libraries.dialogs\_py.MultipleSelectionDialog* method), 171
- option\_readfile () (*robot.libraries.dialogs\_py.PassFailDialog* method), 185
- option\_readfile () (*robot.libraries.dialogs\_py.SelectionDialog* method), 157
- OrTagPattern (class in *robot.model.tags*), 242
- Output (class in *robot.output.output*), 276
- output (*robot.conf.settings.RebotSettings* attribute), 33
- output (*robot.conf.settings.RobotSettings* attribute), 32
- output () (*robot.output.console.verbose.VerboseWriter* method), 269
- output\_directory (*robot.conf.settings.RebotSettings* attribute), 33
- output\_directory (*robot.conf.settings.RobotSettings* attribute), 32
- output\_file () (*robot.output.console.dotted.DottedOutput* method), 264
- output\_file () (*robot.output.console.verbose.VerboseOutput* method), 269
- output\_file () (*robot.output.filelogger.FileLogger* method), 270
- output\_file () (*robot.output.listeners.LibraryListeners* method), 272
- output\_file () (*robot.output.listeners.Listeners* method), 272
- output\_file () (*robot.output.logger.Logger* method), 273
- OutputCapturer (class in *robot.running.outputcapture*), 540
- OutputWriter (class in *robot.reporting.outputwriter*), 359
- overrideredirect () (*robot.libraries.dialogs\_py.InputDialog* method), 143
- overrideredirect () (*robot.libraries.dialogs\_py.MessageDialog* method), 129

method), 129  
 overriddenirect ()  
   (robot.libraries.dialogs\_py.MultipleSelectionDialog  
   method), 171  
 overriddenirect ()  
   (robot.libraries.dialogs\_py.PassFailDialog  
   method), 185  
 overriddenirect ()  
   (robot.libraries.dialogs\_py.SelectionDialog  
   method), 157

## P

pack\_propagate () (robot.libraries.dialogs\_py.InputDialog  
   method), 143  
 pack\_propagate () (robot.libraries.dialogs\_py.MessageDialog  
   method), 129  
 pack\_propagate () (robot.libraries.dialogs\_py.MultipleSelectionDialog  
   method), 171  
 pack\_propagate () (robot.libraries.dialogs\_py.PassFailDialog  
   method), 185  
 pack\_propagate () (robot.libraries.dialogs\_py.SelectionDialog  
   method), 157  
 pack\_slaves () (robot.libraries.dialogs\_py.InputDialog  
   method), 143  
 pack\_slaves () (robot.libraries.dialogs\_py.MessageDialog  
   method), 129  
 pack\_slaves () (robot.libraries.dialogs\_py.MultipleSelectionDialog  
   method), 171  
 pack\_slaves () (robot.libraries.dialogs\_py.PassFailDialog  
   method), 185  
 pack\_slaves () (robot.libraries.dialogs\_py.SelectionDialog  
   method), 157  
 ParagraphFormatter (class in  
   robot.utils.htmlformatters), 562  
 parent (robot.model.body.BodyItem attribute), 194  
 parent (robot.model.control.Break attribute), 215  
 parent (robot.model.control.Continue attribute), 213  
 parent (robot.model.control.For attribute), 204  
 parent (robot.model.control.If attribute), 207  
 parent (robot.model.control.IfBranch attribute), 206  
 parent (robot.model.control.Return attribute), 211  
 parent (robot.model.control.Try attribute), 210  
 parent (robot.model.control.TryBranch attribute), 209  
 parent (robot.model.control.While attribute), 205  
 parent (robot.model.keyword.Keyword attribute), 224  
 parent (robot.model.message.Message attribute), 227  
 parent (robot.model.testcase.TestCase attribute), 248  
 parent (robot.model.testsuite.TestSuite attribute), 250  
 parent (robot.output.loggerhelper.Message attribute),  
   276  
 parent (robot.result.model.Break attribute), 453  
 parent (robot.result.model.Continue attribute), 451  
 parent (robot.result.model.For attribute), 433  
 parent (robot.result.model.ForIteration attribute), 429  
 parent (robot.result.model.If attribute), 442  
 parent (robot.result.model.IfBranch attribute), 440  
 parent (robot.result.model.Keyword attribute), 456  
 parent (robot.result.model.Message attribute), 428  
 parent (robot.result.model.Return attribute), 449  
 parent (robot.result.model.TestCase attribute), 459  
 parent (robot.result.model.TestSuite attribute), 461  
 parent (robot.result.model.Try attribute), 446  
 parent (robot.result.model.TryBranch attribute), 444  
 parent (robot.result.model.While attribute), 437  
 parent (robot.result.model.WhileIteration attribute),  
   433  
 parent (robot.running.model.Break attribute), 532  
 parent (robot.running.model.Continue attribute), 531  
 parent (robot.running.model.For attribute), 521  
 parent (robot.running.model.If attribute), 525  
 parent (robot.running.model.IfBranch attribute), 524  
 parent (robot.running.model.Keyword attribute), 519  
 parent (robot.running.model.Return attribute), 529  
 parent (robot.running.model.TestCase attribute), 533  
 parent (robot.running.model.TestSuite attribute), 537  
 parent (robot.running.model.Try attribute), 528  
 parent (robot.running.model.TryBranch attribute), 527  
 parent (robot.running.model.While attribute), 523  
 ParentMessage (class in robot.running.status), 546  
 parse () (robot.parsing.parser.blockparsers.BlockParser  
   method), 353  
 parse () (robot.parsing.parser.blockparsers.ForParser  
   method), 354  
 parse () (robot.parsing.parser.blockparsers.IfParser  
   method), 354  
 parse () (robot.parsing.parser.blockparsers.KeywordParser  
   method), 354  
 parse () (robot.parsing.parser.blockparsers.NestedBlockParser  
   method), 354  
 parse () (robot.parsing.parser.blockparsers.Parser  
   method), 353  
 parse () (robot.parsing.parser.blockparsers.TestCaseParser  
   method), 354  
 parse () (robot.parsing.parser.blockparsers.TryParser  
   method), 354  
 parse () (robot.parsing.parser.blockparsers.WhileParser  
   method), 354  
 parse () (robot.parsing.parser.fileparser.CommentSectionParser  
   method), 355  
 parse () (robot.parsing.parser.fileparser.FileParser  
   method), 354  
 parse () (robot.parsing.parser.fileparser.ImplicitCommentSectionParser  
   method), 355  
 parse () (robot.parsing.parser.fileparser.KeywordSectionParser  
   method), 355  
 parse () (robot.parsing.parser.fileparser.SectionParser  
   method), 355  
 parse () (robot.parsing.parser.fileparser.SettingSectionParser



- [method\), 355](#)
- [parse\(\) \(robot.parsing.parser.fileparser.TestCaseSectionParser method\), 355](#)
- [parse\(\) \(robot.parsing.parser.fileparser.VariableSectionParser method\), 355](#)
- [parse\(\) \(robot.running.arguments.argumentparser.DynamicArgumentParser method\), 493](#)
- [parse\(\) \(robot.running.arguments.argumentparser.PythonArgumentParser method\), 493](#)
- [parse\(\) \(robot.running.arguments.argumentparser.UserKeywordArgumentParser method\), 493](#)
- [parse\(\) \(robot.running.arguments.embedded.EmbeddedArgumentParser method\), 495](#)
- [parse\(\) \(robot.running.builder.builders.SuiteStructureParser method\), 504](#)
- [parse\\_args\(\) \(robot.utils.argumentparser.ArgumentParser method\), 554](#)
- [parse\\_arguments\(\) \(robot.libdoc.LibDoc method\), 580](#)
- [parse\\_arguments\(\) \(robot.rebot.Rebot method\), 582](#)
- [parse\\_arguments\(\) \(robot.run.RobotFramework method\), 583](#)
- [parse\\_arguments\(\) \(robot.testdoc.TestDoc method\), 585](#)
- [parse\\_arguments\(\) \(robot.utils.application.Application method\), 553](#)
- [parse\\_init\\_file\(\) \(robot.running.builder.parsers.BaseParser method\), 505](#)
- [parse\\_init\\_file\(\) \(robot.running.builder.parsers.NoInitFileDirectoryParser method\), 505](#)
- [parse\\_init\\_file\(\) \(robot.running.builder.parsers.RestParser method\), 505](#)
- [parse\\_init\\_file\(\) \(robot.running.builder.parsers.RobotParser method\), 505](#)
- [parse\\_resource\\_file\(\) \(robot.running.builder.parsers.BaseParser method\), 505](#)
- [parse\\_resource\\_file\(\) \(robot.running.builder.parsers.NoInitFileDirectoryParser method\), 505](#)
- [parse\\_resource\\_file\(\) \(robot.running.builder.parsers.RestParser method\), 505](#)
- [parse\\_resource\\_file\(\) \(robot.running.builder.parsers.RobotParser method\), 505](#)
- [parse\\_response\(\) \(robot.libraries.Remote.TimeoutHTTPTransport method\), 95](#)
- [parse\\_response\(\) \(robot.libraries.Remote.TimeoutHTTPTransport method\), 95](#)
- [parse\\_suite\\_file\(\) \(robot.running.builder.parsers.BaseParser method\), 505](#)
- [parse\\_suite\\_file\(\) \(robot.running.builder.parsers.NoInitFileDirectoryParser method\), 505](#)
- [parse\\_suite\\_file\(\) \(robot.running.builder.parsers.RestParser method\), 505](#)
- [parse\\_suite\\_file\(\) \(robot.running.builder.parsers.RobotParser method\), 505](#)
- [parse\\_xml\(\) \(robot.libraries.XML.XML method\), 116](#)
- [Parser \(class in robot.parsing.parser.blockparsers\), 353](#)
- [PASS \(robot.result.model.Break attribute\), 452](#)
- [PASS \(robot.result.model.Continue attribute\), 450](#)
- [PASS \(robot.result.model.For attribute\), 431](#)
- [PASS \(robot.result.model.ForIteration attribute\), 429](#)
- [PASS \(robot.result.model.If attribute\), 441](#)
- [PASS \(robot.result.model.IfBranch attribute\), 438](#)
- [PASS \(robot.result.model.Keyword attribute\), 455](#)
- [PASS \(robot.result.model.Return attribute\), 448](#)
- [PASS \(robot.result.model.StatusMixin attribute\), 428](#)
- [PASS \(robot.result.model.TestCase attribute\), 458](#)
- [PASS \(robot.result.model.TestSuite attribute\), 460](#)
- [PASS \(robot.result.model.Try attribute\), 445](#)
- [PASS \(robot.result.model.TryBranch attribute\), 443](#)
- [PASS \(robot.result.model.While attribute\), 436](#)
- [PASS \(robot.result.model.WhileIteration attribute\), 434](#)
- [pass\\_execution\(\) \(robot.libraries.BuiltIn.BuiltIn method\), 50](#)
- [pass\\_execution\\_if\(\) \(robot.libraries.BuiltIn.BuiltIn method\), 50](#)
- [passed \(robot.model.stats.Stat attribute\), 240](#)
- [passed \(robot.model.totalstatistics.TotalStatistics attribute\), 253](#)
- [passed \(robot.result.model.Break attribute\), 453](#)
- [passed \(robot.result.model.Continue attribute\), 451](#)
- [passed \(robot.result.model.For attribute\), 433](#)
- [passed \(robot.result.model.ForIteration attribute\), 430](#)
- [passed \(robot.result.model.If attribute\), 442](#)
- [passed \(robot.result.model.IfBranch attribute\), 440](#)
- [passed \(robot.result.model.Keyword attribute\), 456](#)
- [passed \(robot.result.model.Return attribute\), 449](#)
- [passed \(robot.result.model.StatusMixin attribute\), 428](#)
- [passed \(robot.result.model.TestCase attribute\), 459](#)
- [passed \(robot.result.model.TestSuite attribute\), 460](#)
- [passed \(robot.result.model.Try attribute\), 446](#)
- [passed \(robot.result.model.TryBranch attribute\), 444](#)
- [passed \(robot.result.model.While attribute\), 437](#)

- passed (*robot.result.model.WhileIteration* attribute), 435
- passed (*robot.running.status.SuiteStatus* attribute), 545
- passed (*robot.running.status.TestStatus* attribute), 546
- PassedKeywordRemover (class in *robot.result.keywordremover*), 383
- PassExecution, 578
- PassFailDialog (class in *robot.libraries.dialogs\_py*), 178
- pattern\_type (*robot.model.control.TryBranch* attribute), 209
- pattern\_type (*robot.parsing.model.blocks.Try* attribute), 309
- pattern\_type (*robot.parsing.model.statements.ExceptHeader* attribute), 344
- pattern\_type (*robot.result.model.TryBranch* attribute), 444
- pattern\_type (*robot.running.model.TryBranch* attribute), 527
- PatternHandler (class in *robot.result.xmlelementhandlers*), 486
- patterns (*robot.model.control.TryBranch* attribute), 209
- patterns (*robot.parsing.model.blocks.Try* attribute), 309
- patterns (*robot.parsing.model.statements.ExceptHeader* attribute), 344
- patterns (*robot.result.model.TryBranch* attribute), 444
- patterns (*robot.running.model.TryBranch* attribute), 527
- pause\_execution() (in module *robot.libraries.Dialogs*), 77
- place\_slaves() (*robot.libraries.dialogs\_py.InputDialog* method), 143
- place\_slaves() (*robot.libraries.dialogs\_py.MessageDialog* method), 129
- place\_slaves() (*robot.libraries.dialogs\_py.MultipleSelectionDialog* method), 171
- place\_slaves() (*robot.libraries.dialogs\_py.PassFailDialog* method), 185
- place\_slaves() (*robot.libraries.dialogs\_py.SelectionDialog* method), 157
- plural\_or\_not() (in module *robot.utils.misc*), 565
- pop() (*robot.model.body.BaseBody* method), 195
- pop() (*robot.model.body.Body* method), 197
- pop() (*robot.model.body.Branches* method), 199
- pop() (*robot.model.itemlist.ItemList* method), 224
- pop() (*robot.model.keyword.Keywords* method), 227
- pop() (*robot.model.message.Messages* method), 229
- pop() (*robot.model.metadata.Metadata* method), 229
- pop() (*robot.model.testcase.TestCases* method), 249
- pop() (*robot.model.testsuite.TestSuites* method), 253
- pop() (*robot.result.model.Body* method), 423
- pop() (*robot.result.model.Branches* method), 425
- pop() (*robot.result.model.Iterations* method), 426
- pop() (*robot.running.model.Body* method), 517
- pop() (*robot.running.model.Imports* method), 539
- pop() (*robot.utils.dotdict.DotDict* method), 559
- pop() (*robot.utils.normalizing.NormalizedDict* method), 566
- pop\_from\_dictionary() (*robot.libraries.Collections.Collections* method), 71
- popen\_config(*robot.libraries.Process.ProcessConfiguration* attribute), 93
- popitem() (*robot.model.metadata.Metadata* method), 229
- popitem() (*robot.utils.dotdict.DotDict* method), 559
- popitem() (*robot.utils.normalizing.NormalizedDict* method), 566
- positional (*robot.running.arguments.argumentspec.ArgumentSpec* attribute), 494
- POSITIONAL\_ONLY (*robot.running.arguments.argumentspec.ArgInfo* attribute), 494
- POSITIONAL\_ONLY\_MARKER (*robot.running.arguments.argumentspec.ArgInfo* attribute), 494
- POSITIONAL\_OR\_NAMED (*robot.running.arguments.argumentspec.ArgInfo* attribute), 494
- positionfrom() (*robot.libraries.dialogs\_py.InputDialog* method), 143
- positionfrom() (*robot.libraries.dialogs\_py.MessageDialog* method), 129
- positionfrom() (*robot.libraries.dialogs\_py.MultipleSelectionDialog* method), 171
- positionfrom() (*robot.libraries.dialogs\_py.PassFailDialog* method), 185
- positionfrom() (*robot.libraries.dialogs\_py.SelectionDialog* method), 157
- positional\_modifiers (*robot.conf.settings.RebotSettings* attribute), 33
- pos\_rebot\_modifiers (*robot.conf.settings.RobotSettings* attribute), 32
- pos\_run\_modifiers (*robot.conf.settings.RobotSettings* attribute), 32
- PreformattedFormatter (class in *robot.utils.htmlformatters*), 562
- printable\_name() (in module *robot.utils.misc*), 565
- priority (*robot.running.modelcombiner.ModelCombiner* attribute), 539
- Process (class in *robot.libraries.Process*), 87
- process() (*robot.utils.argumentparser.ArgFileParser* method), 555
- process\_empty\_suite (*robot.conf.settings.RebotSettings* attribute), 33
- process\_rawq() (*robot.libraries.Telnet.TelnetConnection*

- method), 110
- process\_should\_be\_running() (robot.libraries.Process.Process method), 91
- process\_should\_be\_stopped() (robot.libraries.Process.Process method), 91
- ProcessConfiguration (class in robot.libraries.Process), 93
- propagate() (robot.libraries.dialogs\_py.InputDialog method), 143
- propagate() (robot.libraries.dialogs\_py.MessageDialog method), 129
- propagate() (robot.libraries.dialogs\_py.MultipleSelectionDialog method), 171
- propagate() (robot.libraries.dialogs\_py.PassFailDialog method), 185
- propagate() (robot.libraries.dialogs\_py.SelectionDialog method), 157
- protocol() (robot.libraries.dialogs\_py.InputDialog method), 143
- protocol() (robot.libraries.dialogs\_py.MessageDialog method), 129
- protocol() (robot.libraries.dialogs\_py.MultipleSelectionDialog method), 171
- protocol() (robot.libraries.dialogs\_py.PassFailDialog method), 185
- protocol() (robot.libraries.dialogs\_py.SelectionDialog method), 157
- prune\_input() (robot.reporting.jsbuildingcontext.JsBuildingContext method), 358
- py2to3() (in module robot.utils), 553
- py3to2() (in module robot.utils), 553
- PythonArgumentParser (class in robot.running.arguments.argumentparser), 493
- PythonCapturer (class in robot.running.outputcapture), 540
- PythonImporter (class in robot.variables.filesetter), 569
- pythonpath (robot.conf.settings.RebotSettings attribute), 33
- pythonpath (robot.conf.settings.RobotSettings attribute), 32
- Q**
- QuietOutput (class in robot.output.console.quiet), 269
- quit() (robot.libraries.dialogs\_py.InputDialog method), 143
- quit() (robot.libraries.dialogs\_py.MessageDialog method), 129
- quit() (robot.libraries.dialogs\_py.MultipleSelectionDialog method), 171
- quit() (robot.libraries.dialogs\_py.PassFailDialog method), 185
- quit() (robot.libraries.dialogs\_py.SelectionDialog method), 157
- R**
- raise\_deprecation\_error() (robot.model.keyword.Keywords class method), 227
- raise\_error() (robot.utils.connectioncache.NoConnection method), 558
- randomize() (robot.running.model.TestSuite method), 535
- randomize\_seed (robot.conf.settings.RobotSettings attribute), 31
- randomize\_suites (robot.conf.settings.RobotSettings attribute), 31
- randomize\_tests (robot.conf.settings.RobotSettings attribute), 31
- Randomizer (class in robot.running.randomizer), 541
- rawq\_getchar() (robot.libraries.Telnet.TelnetConnection method), 110
- read() (robot.libraries.Telnet.TelnetConnection method), 109
- read() (robot.libraries.Telnet.TerminalEmulator method), 111
- read() (robot.utils.filereader.FileReader method), 561
- read\_all() (robot.libraries.Telnet.TelnetConnection method), 111
- read\_all\_bytes() (robot.libraries.Telnet.TelnetConnection method), 111
- read\_lazy() (robot.libraries.Telnet.TelnetConnection method), 111
- read\_rest\_data() (in module robot.utils), 553
- read\_sb\_data() (robot.libraries.Telnet.TelnetConnection method), 111
- read\_some() (robot.libraries.Telnet.TelnetConnection method), 111
- read\_until() (robot.libraries.Telnet.TelnetConnection method), 109
- read\_until() (robot.libraries.Telnet.TerminalEmulator method), 111
- read\_until\_prompt() (robot.libraries.Telnet.TelnetConnection method), 109
- read\_until\_regexp() (robot.libraries.Telnet.TelnetConnection method), 109
- read\_until\_regexp() (robot.libraries.Telnet.TerminalEmulator method), 111
- read\_very\_eager() (robot.libraries.Telnet.TelnetConnection method), 111

`read_very_lazy()` (*robot.libraries.Telnet.TelnetConnection* class method), 322  
`readlines()` (*robot.utils.filereader.FileReader* class method), 561  
`real` (*robot.reporting.stringcache.StringIndex* attribute), 365  
`Rebot` (class in *robot.rebot*), 582  
`rebot()` (in module *robot*), 9  
`rebot()` (in module *robot.rebot*), 582  
`rebot_cli()` (in module *robot*), 9  
`rebot_cli()` (in module *robot.rebot*), 582  
`RebotSettings` (class in *robot.conf.settings*), 32  
`recommend_similar_keywords()` (*robot.running.namespace.KeywordRecommendationFinder* class method), 540  
`RecommendationFinder` (class in *robot.utils.recommendations*), 567  
`red()` (*robot.output.console.highlighting.AnsiHighlighter* class method), 268  
`red()` (*robot.output.console.highlighting.DosHighlighter* class method), 268  
`red()` (*robot.output.console.highlighting.NoHighlighting* class method), 268  
`regexp_escape()` (*robot.libraries.BuiltIn.BuiltIn* class method), 50  
`register()` (*robot.libraries.dialogs\_py.InputDialog* class method), 143  
`register()` (*robot.libraries.dialogs\_py.MessageDialog* class method), 129  
`register()` (*robot.libraries.dialogs\_py.MultipleSelectionDialog* class method), 171  
`register()` (*robot.libraries.dialogs\_py.PassFailDialog* class method), 185  
`register()` (*robot.libraries.dialogs\_py.SelectionDialog* class method), 157  
`register()` (*robot.model.body.BaseBody* class method), 194  
`register()` (*robot.model.body.Body* class method), 197  
`register()` (*robot.model.body.Branches* class method), 199  
`register()` (*robot.output.listenermethods.LibraryListenerMethods* class method), 272  
`register()` (*robot.output.listeners.LibraryListeners* class method), 272  
`register()` (*robot.parsing.model.statements.Arguments* class method), 334  
`register()` (*robot.parsing.model.statements.Break* class method), 350  
`register()` (*robot.parsing.model.statements.Comment* class method), 351  
`register()` (*robot.parsing.model.statements.Continue* class method), 349  
`register()` (*robot.parsing.model.statements.DefaultTags* class method), 322  
`register()` (*robot.parsing.model.statements.Documentation* class method), 319  
`register()` (*robot.parsing.model.statements.DocumentationOrMetadata* class method), 312  
`register()` (*robot.parsing.model.statements.ElseHeader* class method), 342  
`register()` (*robot.parsing.model.statements.ElseIfHeader* class method), 341  
`register()` (*robot.parsing.model.statements.EmptyLine* class method), 352  
`register()` (*robot.parsing.model.statements.End* class method), 346  
`register()` (*robot.parsing.model.statements.Error* class method), 352  
`register()` (*robot.parsing.model.statements.ExceptHeader* class method), 344  
`register()` (*robot.parsing.model.statements.FinallyHeader* class method), 345  
`register()` (*robot.parsing.model.statements.Fixture* class method), 315  
`register()` (*robot.parsing.model.statements.ForceTags* class method), 321  
`register()` (*robot.parsing.model.statements.ForHeader* class method), 337  
`register()` (*robot.parsing.model.statements.IfElseHeader* class method), 338  
`register()` (*robot.parsing.model.statements.IfHeader* class method), 339  
`register()` (*robot.parsing.model.statements.InlineIfHeader* class method), 340  
`register()` (*robot.parsing.model.statements.KeywordCall* class method), 336  
`register()` (*robot.parsing.model.statements.KeywordName* class method), 329  
`register()` (*robot.parsing.model.statements.LibraryImport* class method), 317  
`register()` (*robot.parsing.model.statements.LoopControl* class method), 348  
`register()` (*robot.parsing.model.statements.Metadata* class method), 320  
`register()` (*robot.parsing.model.statements.MultiValue* class method), 314  
`register()` (*robot.parsing.model.statements.NoArgumentHeader* class method), 342  
`register()` (*robot.parsing.model.statements.ResourceImport* class method), 317  
`register()` (*robot.parsing.model.statements.Return* class method), 335  
`register()` (*robot.parsing.model.statements.ReturnStatement* class method), 347  
`register()` (*robot.parsing.model.statements.SectionHeader* class method), 316  
`register()` (*robot.parsing.model.statements.Setup* class method), 317



class method), 330  
 register() (robot.parsing.model.statements.SingleValueStatement class method), 313  
 register() (robot.parsing.model.statements.Statement class method), 311  
 register() (robot.parsing.model.statements.SuiteSetup class method), 322  
 register() (robot.parsing.model.statements.SuiteTeardown class method), 323  
 register() (robot.parsing.model.statements.Tags class method), 332  
 register() (robot.parsing.model.statements.Teardown class method), 331  
 register() (robot.parsing.model.statements.Template class method), 332  
 register() (robot.parsing.model.statements.TemplateArgument class method), 337  
 register() (robot.parsing.model.statements.TestCaseName class method), 328  
 register() (robot.parsing.model.statements.TestSetup class method), 324  
 register() (robot.parsing.model.statements.TestTeardown class method), 325  
 register() (robot.parsing.model.statements.TestTemplate class method), 326  
 register() (robot.parsing.model.statements.TestTimeout class method), 327  
 register() (robot.parsing.model.statements.Timeout class method), 333  
 register() (robot.parsing.model.statements.TryHeader class method), 343  
 register() (robot.parsing.model.statements.Variable class method), 328  
 register() (robot.parsing.model.statements.VariablesImport class method), 318  
 register() (robot.parsing.model.statements.WhileHeader class method), 347  
 register() (robot.result.model.Body class method), 423  
 register() (robot.result.model.Branches class method), 425  
 register() (robot.result.model.Iterations class method), 426  
 register() (robot.result.xml\_element\_handlers.ArgumentHandler class method), 490  
 register() (robot.result.xml\_element\_handlers.ArgumentsHandler class method), 489  
 register() (robot.result.xml\_element\_handlers.AssignHandler class method), 489  
 register() (robot.result.xml\_element\_handlers.BranchHandler class method), 485  
 register() (robot.result.xml\_element\_handlers.BreakHandler class method), 486  
 register() (robot.result.xml\_element\_handlers.ContinueHandler class method), 486  
 register() (robot.result.xml\_element\_handlers.DocHandler class method), 487  
 register() (robot.result.xml\_element\_handlers.ElementHandler class method), 483  
 register() (robot.result.xml\_element\_handlers.ErrorMessageHandler class method), 490  
 register() (robot.result.xml\_element\_handlers.ErrorsHandler class method), 490  
 register() (robot.result.xml\_element\_handlers.ForHandler class method), 484  
 register() (robot.result.xml\_element\_handlers.IfHandler class method), 485  
 register() (robot.result.xml\_element\_handlers.IterationHandler class method), 485  
 register() (robot.result.xml\_element\_handlers.KeywordHandler class method), 484  
 register() (robot.result.xml\_element\_handlers.MessageHandler class method), 487  
 register() (robot.result.xml\_element\_handlers.MetadataHandler class method), 487  
 register() (robot.result.xml\_element\_handlers.MetadataItemHandler class method), 488  
 register() (robot.result.xml\_element\_handlers.MetaHandler class method), 488  
 register() (robot.result.xml\_element\_handlers.PatternHandler class method), 486  
 register() (robot.result.xml\_element\_handlers.ReturnHandler class method), 486  
 register() (robot.result.xml\_element\_handlers.RobotHandler class method), 483  
 register() (robot.result.xml\_element\_handlers.RootHandler class method), 483  
 register() (robot.result.xml\_element\_handlers.StatisticsHandler class method), 491  
 register() (robot.result.xml\_element\_handlers.StatusHandler class method), 487  
 register() (robot.result.xml\_element\_handlers.SuiteHandler class method), 484  
 register() (robot.result.xml\_element\_handlers.TagHandler class method), 488  
 register() (robot.result.xml\_element\_handlers.TagsHandler class method), 488  
 register() (robot.result.xml\_element\_handlers.TestHandler class method), 484  
 register() (robot.result.xml\_element\_handlers.TimeoutHandler class method), 489  
 register() (robot.result.xml\_element\_handlers.TryHandler class method), 486  
 register() (robot.result.xml\_element\_handlers.ValueHandler class method), 490  
 register() (robot.result.xml\_element\_handlers.VarHandler class method), 489  
 register() (robot.result.xml\_element\_handlers.WhileHandler class method), 486

[class method](#)), 485  
[register\(\)](#) ([robot.running.arguments.typeconverters.BooleanConverter](#) class method), 497  
[register\(\)](#) ([robot.running.arguments.typeconverters.BytesArgumentConverter](#) class method), 499  
[register\(\)](#) ([robot.running.arguments.typeconverters.BytesConverter](#) class method), 498  
[register\(\)](#) ([robot.running.arguments.typeconverters.CombinedConverter](#) class method), 503  
[register\(\)](#) ([robot.running.arguments.typeconverters.CustomConverter](#) class method), 503  
[register\(\)](#) ([robot.running.arguments.typeconverters.DateConverter](#) class method), 499  
[register\(\)](#) ([robot.running.arguments.typeconverters.DateTimeConverter](#) class method), 499  
[register\(\)](#) ([robot.running.arguments.typeconverters.DecimalConverter](#) class method), 498  
[register\(\)](#) ([robot.running.arguments.typeconverters.DictionaryConverter](#) class method), 501  
[register\(\)](#) ([robot.running.arguments.typeconverters.EnumConverter](#) class method), 496  
[register\(\)](#) ([robot.running.arguments.typeconverters.FloatConverter](#) class method), 497  
[register\(\)](#) ([robot.running.arguments.typeconverters.FrozenSetConverter](#) class method), 502  
[register\(\)](#) ([robot.running.arguments.typeconverters.IntegerConverter](#) class method), 497  
[register\(\)](#) ([robot.running.arguments.typeconverters.ListConverter](#) class method), 501  
[register\(\)](#) ([robot.running.arguments.typeconverters.NoneConverter](#) class method), 500  
[register\(\)](#) ([robot.running.arguments.typeconverters.SetConverter](#) class method), 502  
[register\(\)](#) ([robot.running.arguments.typeconverters.StringConverter](#) class method), 496  
[register\(\)](#) ([robot.running.arguments.typeconverters.TimeDeltaConverter](#) class method), 500  
[register\(\)](#) ([robot.running.arguments.typeconverters.TypeConverter](#) class method), 501  
[register\(\)](#) ([robot.running.arguments.typeconverters.TypedConverter](#) class method), 495  
[register\(\)](#) ([robot.running.model.Body](#) class method), 517  
[register\(\)](#) ([robot.utils.connectioncache.ConnectionCache](#) class method), 558  
[register\\_console\\_logger\(\)](#) ([robot.output.logger.Logger](#) class method), 273  
[register\\_error\\_listener\(\)](#) ([robot.output.logger.Logger](#) class method), 273  
[register\\_error\\_listener\(\)](#) ([robot.output.output.Output](#) class method), 276  
[register\\_listeners\(\)](#) ([robot.output.logger.Logger](#) class method), 273  
[register\\_logger\(\)](#) ([robot.output.logger.Logger](#) class method), 273  
[method](#)), 273  
[run\\_keyword\(\)](#) ([robot.libraries.BuiltIn](#) class method), 65  
[log\(\)](#) ([robot.output.logger.Logger](#) class method), 273  
[xml\\_logger\(\)](#) ([robot.output.logger.Logger](#) class method), 273  
[source\(\)](#) ([robot.reporting.jsbuildingcontext JsBuildingContext](#) class method), 357  
[release\(\)](#) ([robot.output.pyloggingconf.RobotHandler](#) class method), 278  
[release\(\)](#) ([robot.running.outputcapture.PythonCapturer](#) class method), 540  
[reload\\_library\(\)](#) ([robot.libraries.BuiltIn.BuiltIn](#) class method), 50  
[reload\\_library\(\)](#) ([robot.running.namespace.Namespace](#) class method), 540  
[Remote](#) (class in [robot.libraries.Remote](#)), 94  
[RemoteError](#), 579  
[RemoteResult](#) (class in [robot.libraries.Remote](#)), 94  
[Message](#) (class in [robot.result.keywordremover](#)), 413  
[robot.model.body.BaseBody](#) (class in [robot.model.body](#)), 195  
[remove\(\)](#) ([robot.model.body.Body](#) class method), 197  
[remove\(\)](#) ([robot.model.body.Branches](#) class method), 199  
[remove\(\)](#) ([robot.model.itemlist.ItemList](#) class method), 224  
[remove\(\)](#) ([robot.model.keyword.Keywords](#) class method), 227  
[remove\(\)](#) ([robot.model.message.Messages](#) class method), 229  
[remove\(\)](#) ([robot.model.tags.Tags](#) class method), 242  
[remove\(\)](#) ([robot.model.testcase.TestCases](#) class method), 249  
[remove\(\)](#) ([robot.model.testsuite.TestSuites](#) class method), 252  
[remove\(\)](#) ([robot.result.model.Body](#) class method), 423  
[remove\(\)](#) ([robot.result.model.Branches](#) class method), 425  
[remove\(\)](#) ([robot.result.model.Iterations](#) class method), 426  
[remove\(\)](#) ([robot.running.model.Body](#) class method), 517  
[remove\(\)](#) ([robot.running.model.Imports](#) class method), 539  
[remove\\_data\\_not\\_needed\\_in\\_report\(\)](#) ([robot.reporting.jsexecutionresult JsExecutionResult](#) class method), 358  
[remove\\_directory\(\)](#) ([robot.libraries.OperatingSystem.OperatingSystem](#) class method), 83  
[remove\\_duplicates\(\)](#) ([robot.libraries.Collections.Collections](#) class method), 71  
[remove\\_element\(\)](#) ([robot.libraries.XML.XML](#) class method), 121  
[remove\\_element\\_attribute\(\)](#) ([robot.libraries.XML.XML](#) class method), 120

`remove_element_attributes()` (*robot.libraries.XML.XML method*), 120  
`remove_elements()` (*robot.libraries.XML.XML method*), 121  
`remove_elements_attribute()` (*robot.libraries.XML.XML method*), 120  
`remove_elements_attributes()` (*robot.libraries.XML.XML method*), 120  
`remove_empty_suites()` (*robot.model.testsuite.TestSuite method*), 252  
`remove_empty_suites()` (*robot.result.model.TestSuite method*), 461  
`remove_empty_suites()` (*robot.running.model.TestSuite method*), 537  
`remove_environment_variable()` (*robot.libraries.OperatingSystem.OperatingSystem method*), 84  
`remove_file()` (*robot.libraries.OperatingSystem.OperatingSystem method*), 82  
`remove_files()` (*robot.libraries.OperatingSystem.OperatingSystem method*), 83  
`remove_from_dictionary()` (*robot.libraries.Collections.Collections method*), 71  
`remove_from_list()` (*robot.libraries.Collections.Collections method*), 71  
`remove_keywords` (*robot.conf.settings.RebotSettings attribute*), 33  
`remove_keywords` (*robot.conf.settings.RobotSettings attribute*), 32  
`remove_keywords()` (*robot.result.model.TestSuite method*), 463  
`remove_path()` (*in module robot.pythonpathsetter*), 581  
`remove_string()` (*robot.libraries.String.String method*), 100  
`remove_string_using_regexp()` (*robot.libraries.String.String method*), 100  
`remove_tags` (*robot.model.configurer.SuiteConfigurer attribute*), 199  
`remove_tags` (*robot.result.configurer.SuiteConfigurer attribute*), 373  
`remove_tags()` (*robot.libraries.BuiltIn.BuiltIn method*), 50  
`remove_values_from_list()` (*robot.libraries.Collections.Collections method*), 71  
`removeFilter()` (*robot.output.pyloggingconf.RobotHandler method*), 278  
`RemoveKeywords` (*class in robot.result.resultbuilder*), 465  
`repeat_keyword()` (*robot.libraries.BuiltIn.BuiltIn method*), 51  
`replace()` (*robot.running.arguments.argumentresolver.VariableReplacer method*), 493  
`replace_defaults()` (*robot.running.arguments.argumentmapper.KeywordCallTemplate method*), 492  
`replace_list()` (*robot.variables.replacer.VariableReplacer method*), 570  
`replace_list()` (*robot.variables.scopes.GlobalVariables method*), 571  
`replace_list()` (*robot.variables.scopes.VariableScopes method*), 571  
`replace_list()` (*robot.variables.variables.Variables method*), 574  
`replace_scalar()` (*robot.variables.replacer.VariableReplacer method*), 570  
`replace_scalar()` (*robot.variables.scopes.GlobalVariables method*), 571  
`replace_scalar()` (*robot.variables.scopes.VariableScopes method*), 571  
`replace_scalar()` (*robot.variables.variables.Variables method*), 574  
`replace_string()` (*robot.libraries.String.String method*), 100  
`replace_string()` (*robot.variables.replacer.VariableReplacer method*), 570  
`replace_string()` (*robot.variables.scopes.GlobalVariables method*), 571  
`replace_string()` (*robot.variables.scopes.VariableScopes method*), 571  
`replace_string()` (*robot.variables.variables.Variables method*), 574  
`replace_string_using_regexp()` (*robot.libraries.String.String method*), 100  
`replace_variables()` (*robot.libraries.BuiltIn.BuiltIn method*), 51  
`replace_variables()` (*robot.running.timeouts.KeywordTimeout method*), 510  
`replace_variables()` (*robot.running.timeouts.TestTimeout method*), 510  
`report` (*robot.conf.settings.RebotSettings attribute*), 33  
`report` (*robot.conf.settings.RobotSettings attribute*), 32  
`report()` (*robot.output.console.dotted.StatusReporter method*), 264  
`report_config` (*robot.conf.settings.RebotSettings attribute*), 33  
`report_error()` (*robot.variables.resolvable.GlobalVariableValue method*), 570  
`report_error()` (*robot.variables.resolvable.Resolvable method*), 570

[report\\_error\(\) \(robot.variables.tablesetter.DictVariableTableValue method\), 574](#)  
[report\\_error\(\) \(robot.variables.tablesetter.ListVariableTableValue method\), 573](#)  
[report\\_error\(\) \(robot.variables.tablesetter.ScalarVariableTableValue method\), 573](#)  
[report\\_error\(\) \(robot.variables.tablesetter.VariableTableValueBase method\), 573](#)  
[report\\_invalid\\_syntax\(\) \(robot.running.model.Import method\), 539](#)  
[report\\_invalid\\_syntax\(\) \(robot.running.model.Variable method\), 538](#)  
[ReportWriter \(class in robot.reporting.logreportwriters\), 359](#)  
[repr\\_args \(robot.model.body.BodyItem attribute\), 194](#)  
[repr\\_args \(robot.model.control.Break attribute\), 215](#)  
[repr\\_args \(robot.model.control.Continue attribute\), 214](#)  
[repr\\_args \(robot.model.control.For attribute\), 203](#)  
[repr\\_args \(robot.model.control.If attribute\), 209](#)  
[repr\\_args \(robot.model.control.IfBranch attribute\), 206](#)  
[repr\\_args \(robot.model.control.Return attribute\), 211](#)  
[repr\\_args \(robot.model.control.Try attribute\), 211](#)  
[repr\\_args \(robot.model.control.TryBranch attribute\), 209](#)  
[repr\\_args \(robot.model.control.While attribute\), 205](#)  
[repr\\_args \(robot.model.keyword.Keyword attribute\), 224](#)  
[repr\\_args \(robot.model.message.Message attribute\), 227](#)  
[repr\\_args \(robot.model.modelobject.ModelObject attribute\), 230](#)  
[repr\\_args \(robot.model.testcase.TestCase attribute\), 247](#)  
[repr\\_args \(robot.model.testsuite.TestSuite attribute\), 250](#)  
[repr\\_args \(robot.output.loggerhelper.Message attribute\), 276](#)  
[repr\\_args \(robot.result.model.Break attribute\), 453](#)  
[repr\\_args \(robot.result.model.Continue attribute\), 451](#)  
[repr\\_args \(robot.result.model.For attribute\), 433](#)  
[repr\\_args \(robot.result.model.ForIteration attribute\), 429](#)  
[repr\\_args \(robot.result.model.If attribute\), 442](#)  
[repr\\_args \(robot.result.model.IfBranch attribute\), 440](#)  
[repr\\_args \(robot.result.model.Keyword attribute\), 456](#)  
[repr\\_args \(robot.result.model.Message attribute\), 428](#)  
[repr\\_args \(robot.result.model.Return attribute\), 449](#)  
[repr\\_args \(robot.result.model.TestCase attribute\), 459](#)  
[repr\\_args \(robot.result.model.TestSuite attribute\), 461](#)  
[repr\\_args \(robot.result.model.Try attribute\), 446](#)  
[repr\\_args \(robot.result.model.TryBranch attribute\), 444](#)  
[repr\\_args \(robot.result.model.While attribute\), 437](#)  
[repr\\_args \(robot.result.model.WhileIteration attribute\), 435](#)  
[repr\\_args \(robot.running.model.Break attribute\), 532](#)  
[repr\\_args \(robot.running.model.Continue attribute\), 531](#)  
[repr\\_args \(robot.running.model.For attribute\), 521](#)  
[repr\\_args \(robot.running.model.If attribute\), 525](#)  
[repr\\_args \(robot.running.model.IfBranch attribute\), 524](#)  
[repr\\_args \(robot.running.model.Keyword attribute\), 519](#)  
[repr\\_args \(robot.running.model.Return attribute\), 529](#)  
[repr\\_args \(robot.running.model.TestCase attribute\), 533](#)  
[repr\\_args \(robot.running.model.TestSuite attribute\), 537](#)  
[repr\\_args \(robot.running.model.Try attribute\), 528](#)  
[repr\\_args \(robot.running.model.TryBranch attribute\), 527](#)  
[repr\\_args \(robot.running.model.While attribute\), 523](#)  
[request\(\) \(robot.libraries.Remote.TimeoutHTTPSTransport method\), 95](#)  
[request\(\) \(robot.libraries.Remote.TimeoutHTTPTransport method\), 95](#)  
[required \(robot.running.arguments.argumentspec.ArgInfo attribute\), 494](#)  
[Reserved \(class in robot.libraries.Reserved\), 95](#)  
[reset\(\) \(robot.output.console.highlighting.AnsiHighlighter method\), 268](#)  
[reset\(\) \(robot.output.console.highlighting.DosHighlighter method\), 269](#)  
[reset\(\) \(robot.output.console.highlighting.NoHighlighting method\), 268](#)  
[reset\(\) \(robot.running.importer.Importer method\), 514](#)  
[reset\\_count\(\) \(robot.output.console.verbose.KeywordMarker method\), 269](#)  
[resizable\(\) \(robot.libraries.dialogs\\_py.InputDialog method\), 143](#)  
[resizable\(\) \(robot.libraries.dialogs\\_py.MessageDialog method\), 130](#)  
[resizable\(\) \(robot.libraries.dialogs\\_py.MultipleSelectionDialog method\), 171](#)  
[resizable\(\) \(robot.libraries.dialogs\\_py.PassFailDialog method\), 185](#)  
[resizable\(\) \(robot.libraries.dialogs\\_py.SelectionDialog method\), 157](#)  
[Resolvable \(class in robot.variables.resolvable\), 570](#)  
[resolve\(\) \(robot.running.arguments.argumentmapper.DefaultValue method\), 493](#)



[resolve\(\)](#) (*robot.running.arguments.argumentresolver.ArgumentResolver* [method](#)), 493  
[resolve\(\)](#) (*robot.running.arguments.argumentresolver.NamedArgumentResolver* [method](#)), 493  
[resolve\(\)](#) (*robot.running.arguments.argumentresolver.NullNamedArgumentResolver* [method](#)), 493  
[resolve\(\)](#) (*robot.running.arguments.argumentspec.ArgumentSpec* [method](#)), 494  
[resolve\(\)](#) (*robot.variables.assigner.NoReturnValueResolver* [method](#)), 568  
[resolve\(\)](#) (*robot.variables.assigner.OneReturnValueResolver* [method](#)), 568  
[resolve\(\)](#) (*robot.variables.assigner.ScalarsAndListReturnValueResolver* [method](#)), 568  
[resolve\(\)](#) (*robot.variables.assigner.ScalarsOnlyReturnValueResolver* [method](#)), 568  
[resolve\(\)](#) (*robot.variables.resolvable.GlobalVariableValue* [method](#)), 570  
[resolve\(\)](#) (*robot.variables.resolvable.ResolvableValue* [method](#)), 570  
[resolve\(\)](#) (*robot.variables.tablesetter.DictVariableTableValue* [method](#)), 574  
[resolve\(\)](#) (*robot.variables.tablesetter.ListVariableTableValue* [method](#)), 573  
[resolve\(\)](#) (*robot.variables.tablesetter.ScalarVariableTableValue* [method](#)), 573  
[resolve\(\)](#) (*robot.variables.tablesetter.VariableTableValue* [method](#)), 573  
[resolve\\_alias\\_or\\_index\(\)](#) (*robot.utils.connectioncache.ConnectionCache* [method](#)), 558  
[resolve\\_arguments\(\)](#) (*robot.running.handlers.EmbeddedArgumentsHandler* [method](#)), 514  
[resolve\\_base\(\)](#) (*robot.variables.search.VariableMatch* [method](#)), 572  
[resolve\\_delayed\(\)](#) (*robot.variables.scopes.GlobalVariables* [method](#)), 571  
[resolve\\_delayed\(\)](#) (*robot.variables.scopes.VariableScopes* [method](#)), 571  
[resolve\\_delayed\(\)](#) (*robot.variables.store.VariableStore* [method](#)), 573  
[resolve\\_delayed\(\)](#) (*robot.variables.variables.Variables* [method](#)), 574  
[resolve\\_delayed\\_message\(\)](#) (*robot.output.loggerhelper.Message* [method](#)), 274  
[RESOURCE](#) (*robot.parsing.lexer.tokens.END* [attribute](#)), 304  
[RESOURCE](#) (*robot.parsing.lexer.tokens.EOS* [attribute](#)), 304

[RESOURCE](#) (*robot.parsing.lexer.tokens.Token* [attribute](#)), 304  
[resource](#) (*robot.running.model.TestSuite* [attribute](#)), 524  
[resource\(\)](#) (*robot.running.model.Imports* [method](#)), 534  
[RESOURCE\\_FILE\\_TYPE](#) (*robot.running.handlerstore.HandlerStore* [attribute](#)), 514  
[RESOURCE\\_FILE\\_TYPE](#) (*robot.running.userkeyword.UserLibrary* [attribute](#)), 552  
[ResourceBuilder](#) (*class* [in](#) *robot.running.builder.transformers*), 507  
[ResourceDocBuilder](#) (*class* [in](#) *robot.libdocpkg.robotbuilder*), 38  
[ResourceFile](#) (*class* [in](#) *robot.running.model*), 538  
[ResourceFileBuilder](#) (*class* [in](#) *robot.running.builder.builders*), 504  
[ResourceFileContext](#) (*class* [in](#) *robot.parsing.lexer.context*), 289  
[ResourceFileSections](#) (*class* [in](#) *robot.parsing.lexer.sections*), 291  
[ResourceFileSettings](#) (*class* [in](#) *robot.parsing.lexer.settings*), 293  
[ResourceImport](#) (*class* [in](#) *robot.parsing.model.statements*), 317  
[RestParser](#) (*class* [in](#) *robot.running.builder.parsers*), 505  
[Result](#) (*class* [in](#) *robot.result.executionresult*), 377  
[result](#) (*robot.reporting.resultwriter.Results* [attribute](#)), 365  
[result](#) (*robot.running.modelcombiner.ModelCombiner* [attribute](#)), 539  
[result\\_config](#) (*robot.libraries.Process.ProcessConfiguration* [attribute](#)), 93  
[Results](#) (*class* [in](#) *robot.reporting.resultwriter*), 365  
[ResultVisitor](#) (*class* [in](#) *robot.result.visitor*), 477  
[ResultWriter](#) (*class* [in](#) *robot.reporting.resultwriter*), 364  
[Return](#) (*class* [in](#) *robot.model.control*), 211  
[Return](#) (*class* [in](#) *robot.parsing.model.statements*), 334  
[Return](#) (*class* [in](#) *robot.result.model*), 447  
[Return](#) (*class* [in](#) *robot.running.model*), 528  
[RETURN](#) (*robot.model.body.BodyItem* [attribute](#)), 193  
[RETURN](#) (*robot.model.control.Break* [attribute](#)), 214  
[RETURN](#) (*robot.model.control.Continue* [attribute](#)), 213  
[RETURN](#) (*robot.model.control.For* [attribute](#)), 204  
[RETURN](#) (*robot.model.control.If* [attribute](#)), 208  
[RETURN](#) (*robot.model.control.IfBranch* [attribute](#)), 207  
[RETURN](#) (*robot.model.control.Return* [attribute](#)), 212  
[RETURN](#) (*robot.model.control.Try* [attribute](#)), 211  
[RETURN](#) (*robot.model.control.TryBranch* [attribute](#)), 209

- RETURN (*robot.model.control.While* attribute), 205
- RETURN (*robot.model.keyword.Keyword* attribute), 225
- RETURN (*robot.model.message.Message* attribute), 228
- RETURN (*robot.output.loggerhelper.Message* attribute), 275
- RETURN (*robot.parsing.lexer.tokens.END* attribute), 304
- RETURN (*robot.parsing.lexer.tokens.EOS* attribute), 302
- RETURN (*robot.parsing.lexer.tokens.Token* attribute), 300
- RETURN (*robot.result.model.Break* attribute), 452
- RETURN (*robot.result.model.Continue* attribute), 450
- RETURN (*robot.result.model.For* attribute), 431
- RETURN (*robot.result.model.ForIteration* attribute), 429
- RETURN (*robot.result.model.If* attribute), 441
- RETURN (*robot.result.model.IfBranch* attribute), 438
- RETURN (*robot.result.model.Keyword* attribute), 455
- RETURN (*robot.result.model.Message* attribute), 427
- RETURN (*robot.result.model.Return* attribute), 448
- RETURN (*robot.result.model.Try* attribute), 445
- RETURN (*robot.result.model.TryBranch* attribute), 443
- RETURN (*robot.result.model.While* attribute), 436
- RETURN (*robot.result.model.WhileIteration* attribute), 434
- RETURN (*robot.running.model.Break* attribute), 531
- RETURN (*robot.running.model.Continue* attribute), 530
- RETURN (*robot.running.model.For* attribute), 520
- RETURN (*robot.running.model.If* attribute), 525
- RETURN (*robot.running.model.IfBranch* attribute), 523
- RETURN (*robot.running.model.Keyword* attribute), 518
- RETURN (*robot.running.model.Return* attribute), 529
- RETURN (*robot.running.model.Try* attribute), 527
- RETURN (*robot.running.model.TryBranch* attribute), 526
- RETURN (*robot.running.model.While* attribute), 522
- return\_class (*robot.model.body.BaseBody* attribute), 194
- return\_class (*robot.model.body.Body* attribute), 197
- return\_class (*robot.model.body.Branches* attribute), 199
- return\_class (*robot.result.model.Body* attribute), 423
- return\_class (*robot.result.model.Branches* attribute), 425
- return\_class (*robot.result.model.Iterations* attribute), 426
- return\_class (*robot.running.model.Body* attribute), 517
- return\_code (*robot.result.executionresult.CombinedResult* attribute), 378
- return\_code (*robot.result.executionresult.Result* attribute), 377
- return\_from\_keyword()
  - (*robot.libraries.BuiltIn.BuiltIn* method), 51
- return\_from\_keyword\_if()
  - (*robot.libraries.BuiltIn.BuiltIn* method), 52
- RETURN\_SETTING (*robot.parsing.lexer.tokens.END* attribute), 304
- RETURN\_SETTING (*robot.parsing.lexer.tokens.EOS* attribute), 302
- RETURN\_SETTING (*robot.parsing.lexer.tokens.Token* attribute), 300
- RETURN\_STATEMENT (*robot.parsing.lexer.tokens.END* attribute), 305
- RETURN\_STATEMENT (*robot.parsing.lexer.tokens.EOS* attribute), 302
- RETURN\_STATEMENT (*robot.parsing.lexer.tokens.Token* attribute), 300
- ReturnFromKeyword, 579
- ReturnHandler (class in *robot.result.xmllelementhandlers*), 486
- ReturnLexer (class in *robot.parsing.lexer.statementlexers*), 298
- ReturnStatement (class in *robot.parsing.model.statements*), 347
- ReturnValueResolver() (in module *robot.variables.assigner*), 568
- reverse() (*robot.model.body.BaseBody* method), 195
- reverse() (*robot.model.body.Body* method), 197
- reverse() (*robot.model.body.Branches* method), 199
- reverse() (*robot.model.itemlist.ItemList* method), 224
- reverse() (*robot.model.keyword.Keywords* method), 227
- reverse() (*robot.model.message.Messages* method), 229
- reverse() (*robot.model.testcase.TestCases* method), 250
- reverse() (*robot.model.testsuite.TestSuites* method), 253
- reverse() (*robot.result.model.Body* method), 423
- reverse() (*robot.result.model.Branches* method), 425
- reverse() (*robot.result.model.Iterations* method), 426
- reverse() (*robot.running.model.Body* method), 517
- reverse() (*robot.running.model.Imports* method), 539
- reverse\_list() (*robot.libraries.Collections.Collections* method), 71
- robot (module), 7
- robot.api (module), 5, 10
- robot.api.deco (module), 10
- robot.api.exceptions (module), 12
- robot.api.logger (module), 14
- robot.api.parsing (module), 15
- robot.conf (module), 22
- robot.conf.gatherfailed (module), 23
- robot.conf.settings (module), 31
- robot.errors (module), 574
- robot.htmldata (module), 33
- robot.htmldata.htmlfilewriter (module), 33
- robot.htmldata.jsonwriter (module), 34

robot.htmldata.template (module), 35  
 robot.libdoc (module), 580  
 robot.libdocpkg (module), 35  
 robot.libdocpkg.builder (module), 35  
 robot.libdocpkg.consoleviewer (module), 36  
 robot.libdocpkg.datatypes (module), 36  
 robot.libdocpkg.htmlutils (module), 36  
 robot.libdocpkg.htmlwriter (module), 37  
 robot.libdocpkg.jsonbuilder (module), 37  
 robot.libdocpkg.jsonwriter (module), 37  
 robot.libdocpkg.model (module), 37  
 robot.libdocpkg.output (module), 38  
 robot.libdocpkg.robotbuilder (module), 38  
 robot.libdocpkg.standardtypes (module), 38  
 robot.libdocpkg.writer (module), 38  
 robot.libdocpkg.xmlbuilder (module), 38  
 robot.libdocpkg.xmlwriter (module), 39  
 robot.libraries (module), 39  
 robot.libraries.BuiltIn (module), 39  
 robot.libraries.Collections (module), 65  
 robot.libraries.DateTime (module), 72  
 robot.libraries.Dialogs (module), 77  
 robot.libraries.dialogs\_py (module), 123  
 robot.libraries.Easter (module), 77  
 robot.libraries.OperatingSystem (module), 78  
 robot.libraries.Process (module), 87  
 robot.libraries.Remote (module), 94  
 robot.libraries.Reserved (module), 95  
 robot.libraries.Screenshot (module), 95  
 robot.libraries.String (module), 97  
 robot.libraries.Telnet (module), 103  
 robot.libraries.XML (module), 112  
 robot.model (module), 193  
 robot.model.body (module), 193  
 robot.model.configurer (module), 199  
 robot.model.control (module), 203  
 robot.model.filter (module), 215  
 robot.model.fixture (module), 223  
 robot.model.itemlist (module), 224  
 robot.model.keyword (module), 224  
 robot.model.message (module), 227  
 robot.model.metadata (module), 229  
 robot.model.modelobject (module), 230  
 robot.model.modifier (module), 230  
 robot.model.namepatterns (module), 235  
 robot.model.statistics (module), 235  
 robot.model.stats (module), 239  
 robot.model.sitestatistics (module), 241  
 robot.model.tags (module), 242  
 robot.model.tagsetter (module), 242  
 robot.model.tagstatistics (module), 247  
 robot.model.testcase (module), 247  
 robot.model.testsuite (module), 250  
 robot.model.totalstatistics (module), 253  
 robot.model.visitor (module), 258  
 robot.output (module), 263  
 robot.output.console (module), 263  
 robot.output.console.dotted (module), 263  
 robot.output.console.highlighting (module), 268  
 robot.output.console.quiet (module), 269  
 robot.output.console.verbose (module), 269  
 robot.output.debugfile (module), 270  
 robot.output.filelogger (module), 270  
 robot.output.librarylogger (module), 270  
 robot.output.listenerarguments (module), 271  
 robot.output.listenermethods (module), 272  
 robot.output.listeners (module), 272  
 robot.output.logger (module), 273  
 robot.output.loggerhelper (module), 274  
 robot.output.output (module), 276  
 robot.output.pyloggingconf (module), 277  
 robot.output.stdoutlogsplitter (module), 278  
 robot.output.xmllogger (module), 278  
 robot.parsing (module), 283  
 robot.parsing.lexer (module), 283  
 robot.parsing.lexer.blocklexers (module), 283  
 robot.parsing.lexer.context (module), 288  
 robot.parsing.lexer.lexer (module), 290  
 robot.parsing.lexer.sections (module), 290  
 robot.parsing.lexer.settings (module), 292  
 robot.parsing.lexer.statementlexers (module), 293  
 robot.parsing.lexer.tokenizer (module), 299  
 robot.parsing.lexer.tokens (module), 299  
 robot.parsing.model (module), 306  
 robot.parsing.model.blocks (module), 306  
 robot.parsing.model.statements (module), 311  
 robot.parsing.model.visitor (module), 353  
 robot.parsing.parser (module), 353  
 robot.parsing.parser.blockparsers (module), 353  
 robot.parsing.parser.fileparser (module), 354  
 robot.parsing.parser.parser (module), 356  
 robot.parsing.suitestructure (module), 356  
 robot.pythonpathsetter (module), 581  
 robot.rebot (module), 581  
 robot.reporting (module), 357  
 robot.reporting.expandkeywordmatcher (module), 357

`robot.reporting.jsbuildingcontext` (*module*), 357  
`robot.reporting.jsexecutionresult` (*module*), 358  
`robot.reporting.jsmodelbuilders` (*module*), 358  
`robot.reporting.jswriter` (*module*), 359  
`robot.reporting.logreportwriters` (*module*), 359  
`robot.reporting.outputwriter` (*module*), 359  
`robot.reporting.resultwriter` (*module*), 364  
`robot.reporting.stringcache` (*module*), 365  
`robot.reporting.xunitwriter` (*module*), 366  
`robot.result` (*module*), 371  
`robot.result.configurer` (*module*), 372  
`robot.result.executionerrors` (*module*), 376  
`robot.result.executionresult` (*module*), 377  
`robot.result.flattenkeywordmatcher` (*module*), 379  
`robot.result.keywordremover` (*module*), 379  
`robot.result.merger` (*module*), 413  
`robot.result.messagefilter` (*module*), 417  
`robot.result.model` (*module*), 421  
`robot.result.modeldeprecation` (*module*), 463  
`robot.result.resultbuilder` (*module*), 464  
`robot.result.suite teardown failed` (*module*), 469  
`robot.result.visitor` (*module*), 477  
`robot.result.xmlelementhandlers` (*module*), 483  
`robot.run` (*module*), 583  
`robot.running` (*module*), 491  
`robot.running.arguments` (*module*), 492  
`robot.running.arguments.argumentconverter` (*module*), 492  
`robot.running.arguments.argumentmapper` (*module*), 492  
`robot.running.arguments.argumentparser` (*module*), 493  
`robot.running.arguments.argumentresolver` (*module*), 493  
`robot.running.arguments.argumentspec` (*module*), 494  
`robot.running.arguments.argumentvalidator` (*module*), 494  
`robot.running.arguments.customconverters` (*module*), 495  
`robot.running.arguments.embedded` (*module*), 495  
`robot.running.arguments.typeconverters` (*module*), 495  
`robot.running.arguments.typevalidator` (*module*), 503  
`robot.running.bodyrunner` (*module*), 511  
`robot.running.builder` (*module*), 503  
`robot.running.builder.builders` (*module*), 504  
`robot.running.builder.parsers` (*module*), 505  
`robot.running.builder.testsettings` (*module*), 505  
`robot.running.builder.transformers` (*module*), 506  
`robot.running.context` (*module*), 513  
`robot.running.dynamicmethods` (*module*), 513  
`robot.running.handlers` (*module*), 513  
`robot.running.handlerstore` (*module*), 514  
`robot.running.importer` (*module*), 514  
`robot.running.librarykeywordrunner` (*module*), 514  
`robot.running.libraryscopes` (*module*), 515  
`robot.running.model` (*module*), 516  
`robot.running.modelcombiner` (*module*), 539  
`robot.running.namespace` (*module*), 540  
`robot.running.outputcapture` (*module*), 540  
`robot.running.randomizer` (*module*), 541  
`robot.running.runkwregister` (*module*), 545  
`robot.running.signalhandler` (*module*), 545  
`robot.running.status` (*module*), 545  
`robot.running.statusreporter` (*module*), 547  
`robot.running.suiterunner` (*module*), 547  
`robot.running.testlibraries` (*module*), 551  
`robot.running.timeouts` (*module*), 510  
`robot.running.timeouts.posix` (*module*), 511  
`robot.running.timeouts.windows` (*module*), 511  
`robot.running.usererrorhandler` (*module*), 551  
`robot.running.userkeyword` (*module*), 552  
`robot.running.userkeywordrunner` (*module*), 552  
`robot.testdoc` (*module*), 585  
`robot.utils` (*module*), 553  
`robot.utils.application` (*module*), 553  
`robot.utils.argumentparser` (*module*), 554  
`robot.utils.asserts` (*module*), 555  
`robot.utils.charwidth` (*module*), 557  
`robot.utils.compress` (*module*), 557  
`robot.utils.connectioncache` (*module*), 557  
`robot.utils.dotdict` (*module*), 558  
`robot.utils.encoding` (*module*), 559  
`robot.utils.encodingsniffer` (*module*), 560  
`robot.utils.error` (*module*), 560  
`robot.utils.escaping` (*module*), 560  
`robot.utils.etreewrapper` (*module*), 560  
`robot.utils.filereader` (*module*), 560  
`robot.utils.frange` (*module*), 561



- `robot.utils.htmlformatters` (*module*), 561
- `robot.utils.importer` (*module*), 562
- `robot.utils.markuputils` (*module*), 564
- `robot.utils.markupwriters` (*module*), 564
- `robot.utils.match` (*module*), 565
- `robot.utils.misc` (*module*), 565
- `robot.utils.normalizing` (*module*), 566
- `robot.utils.platform` (*module*), 566
- `robot.utils.recommendations` (*module*), 567
- `robot.variables` (*module*), 567
- `robot.variables.assigner` (*module*), 568
- `robot.variables.evaluation` (*module*), 568
- `robot.variables.filesetter` (*module*), 569
- `robot.variables.finders` (*module*), 569
- `robot.variables.notfound` (*module*), 570
- `robot.variables.replacer` (*module*), 570
- `robot.variables.resolvable` (*module*), 570
- `robot.variables.scopes` (*module*), 570
- `robot.variables.search` (*module*), 572
- `robot.variables.store` (*module*), 573
- `robot.variables.tablesetter` (*module*), 573
- `robot.variables.variables` (*module*), 574
- `robot.version` (*module*), 586
- `ROBOT_CONTINUE_ON_FAILURE` (*robot.api.exceptions.ContinuableFailure* attribute), 13
- `ROBOT_EXIT_ON_FAILURE` (*robot.api.exceptions.FatalError* attribute), 13
- `robot_handler_enabled()` (*in module robot.output.pyloggingconf*), 277
- `ROBOT_LIBRARY_SCOPE` (*robot.libraries.BuiltIn.BuiltIn* attribute), 42
- `ROBOT_LIBRARY_SCOPE` (*robot.libraries.Collections.Collections* attribute), 66
- `ROBOT_LIBRARY_SCOPE` (*robot.libraries.OperatingSystem.OperatingSystem* attribute), 79
- `ROBOT_LIBRARY_SCOPE` (*robot.libraries.Process.Process* attribute), 90
- `ROBOT_LIBRARY_SCOPE` (*robot.libraries.Remote.Remote* attribute), 94
- `ROBOT_LIBRARY_SCOPE` (*robot.libraries.Reserved.Reserved* attribute), 95
- `ROBOT_LIBRARY_SCOPE` (*robot.libraries.Screenshot.Screenshot* attribute), 96
- `ROBOT_LIBRARY_SCOPE` (*robot.libraries.String.String* attribute), 97
- `ROBOT_LIBRARY_SCOPE` (*robot.libraries.Telnet.Telnet* attribute), 106
- `ROBOT_LIBRARY_SCOPE` (*robot.libraries.XML.XML* attribute), 116
- `ROBOT_LIBRARY_VERSION` (*robot.libraries.BuiltIn.BuiltIn* attribute), 42
- `ROBOT_LIBRARY_VERSION` (*robot.libraries.Collections.Collections* attribute), 66
- `ROBOT_LIBRARY_VERSION` (*robot.libraries.OperatingSystem.OperatingSystem* attribute), 79
- `ROBOT_LIBRARY_VERSION` (*robot.libraries.Process.Process* attribute), 90
- `ROBOT_LIBRARY_VERSION` (*robot.libraries.Screenshot.Screenshot* attribute), 96
- `ROBOT_LIBRARY_VERSION` (*robot.libraries.String.String* attribute), 97
- `ROBOT_LIBRARY_VERSION` (*robot.libraries.Telnet.Telnet* attribute), 106
- `ROBOT_LIBRARY_VERSION` (*robot.libraries.XML.XML* attribute), 116
- `ROBOT_SKIP_EXECUTION` (*robot.api.exceptions.SkipExecution* attribute), 14
- `ROBOT_SUPPRESS_NAME` (*robot.api.exceptions.ContinuableFailure* attribute), 13
- `ROBOT_SUPPRESS_NAME` (*robot.api.exceptions.Error* attribute), 13
- `ROBOT_SUPPRESS_NAME` (*robot.api.exceptions.Failure* attribute), 12
- `ROBOT_SUPPRESS_NAME` (*robot.api.exceptions.FatalError* attribute), 13
- `ROBOT_SUPPRESS_NAME` (*robot.api.exceptions.SkipExecution* attribute), 14
- `ROBOT_SUPPRESS_NAME` (*robot.libraries.Telnet.NoMatchError* attribute), 112
- `RobotError`, 574
- `RobotFramework` (*class in robot.run*), 583
- `RobotHandler` (*class in robot.output.pyloggingconf*), 277
- `RobotHandler` (*class in robot.result.xmllelementhandlers*), 483
- `RobotModelWriter` (*class in robot.reporting.logreportwriters*), 359
- `RobotNotRunningError`, 65
- `RobotParser` (*class in robot.running.builder.parsers*),

- 505
- RobotSettings (class in robot.conf.settings), 31
- RootHandler (class in robot.result.xmlélémenthandlers), 483
- rowconfigure() (robot.libraries.dialogs\_py.InputDialog method), 143
- rowconfigure() (robot.libraries.dialogs\_py.MessageDialog method), 130
- rowconfigure() (robot.libraries.dialogs\_py.MultipleSelectionDialog method), 171
- rowconfigure() (robot.libraries.dialogs\_py.PassFailDialog method), 185
- rowconfigure() (robot.libraries.dialogs\_py.SelectionDialog method), 157
- rpa (robot.conf.settings.RebotSettings attribute), 33
- rpa (robot.conf.settings.RobotSettings attribute), 32
- rpa (robot.model.testsuite.TestSuite attribute), 250
- rpa (robot.result.model.TestSuite attribute), 462
- rpa (robot.running.model.TestSuite attribute), 537
- RulerFormatter (class in robot.utils.htmlformatters), 561
- run() (in module robot), 7
- run() (in module robot.run), 584
- run() (robot.libraries.OperatingSystem.OperatingSystem method), 79
- run() (robot.running.bodyrunner.BodyRunner method), 511
- run() (robot.running.bodyrunner.ForInEnumerateRunner method), 512
- run() (robot.running.bodyrunner.ForInRangeRunner method), 511
- run() (robot.running.bodyrunner.ForInRunner method), 511
- run() (robot.running.bodyrunner.ForInZipRunner method), 511
- run() (robot.running.bodyrunner.IfRunner method), 512
- run() (robot.running.bodyrunner.KeywordRunner method), 511
- run() (robot.running.bodyrunner.TryRunner method), 512
- run() (robot.running.bodyrunner.WhileRunner method), 512
- run() (robot.running.librarykeywordrunner.EmbeddedArgumentsRunner method), 515
- run() (robot.running.librarykeywordrunner.LibraryKeywordRunner method), 514
- run() (robot.running.librarykeywordrunner.RunKeywordRunner method), 515
- run() (robot.running.model.Break method), 531
- run() (robot.running.model.Continue method), 530
- run() (robot.running.model.For method), 520
- run() (robot.running.model.If method), 524
- run() (robot.running.model.Keyword method), 518
- run() (robot.running.model.Return method), 528
- run() (robot.running.model.TestSuite method), 535
- run() (robot.running.model.Try method), 527
- run() (robot.running.model.While method), 521
- run() (robot.running.timeouts.KeywordTimeout method), 510
- run() (robot.running.timeouts.TestTimeout method), 510
- run() (robot.running.usererrorhandler.UserErrorHandler method), 552
- run() (robot.running.userkeywordrunner.EmbeddedArgumentsRunner method), 553
- run() (robot.running.userkeywordrunner.UserKeywordRunner method), 552
- run\_and\_return\_rc() (robot.libraries.OperatingSystem.OperatingSystem method), 79
- run\_and\_return\_rc\_and\_output() (robot.libraries.OperatingSystem.OperatingSystem method), 79
- run\_cli() (in module robot), 8
- run\_cli() (in module robot.run), 583
- run\_empty\_suite (robot.conf.settings.RobotSettings attribute), 32
- run\_keyword() (robot.libraries.BuiltIn.BuiltIn method), 52
- run\_keyword() (robot.libraries.Remote.Remote method), 94
- run\_keyword() (robot.libraries.Remote.XmlRpcRemoteClient method), 94
- run\_keyword\_and\_continue\_on\_failure() (robot.libraries.BuiltIn.BuiltIn method), 52
- run\_keyword\_and\_expect\_error() (robot.libraries.BuiltIn.BuiltIn method), 52
- run\_keyword\_and\_ignore\_error() (robot.libraries.BuiltIn.BuiltIn method), 52
- run\_keyword\_and\_return() (robot.libraries.BuiltIn.BuiltIn method), 53
- run\_keyword\_and\_return\_if() (robot.libraries.BuiltIn.BuiltIn method), 53
- run\_keyword\_and\_return\_status() (robot.libraries.BuiltIn.BuiltIn method), 53
- run\_keyword\_and\_warn\_on\_failure() (robot.libraries.BuiltIn.BuiltIn method), 53
- run\_keyword\_if() (robot.libraries.BuiltIn.BuiltIn method), 53
- run\_keyword\_if\_all\_tests\_passed() (robot.libraries.BuiltIn.BuiltIn method), 53

- 54  
 run\_keyword\_if\_any\_tests\_failed()  
     (robot.libraries.BuiltIn.BuiltIn method), 54  
 run\_keyword\_if\_test\_failed()  
     (robot.libraries.BuiltIn.BuiltIn method), 54  
 run\_keyword\_if\_test\_passed()  
     (robot.libraries.BuiltIn.BuiltIn method), 54  
 run\_keyword\_if\_timeout\_occurred()  
     (robot.libraries.BuiltIn.BuiltIn method), 54  
 run\_keyword\_unless()  
     (robot.libraries.BuiltIn.BuiltIn method), 54  
 run\_keyword\_variant() (in module  
     robot.libraries.BuiltIn), 39  
 run\_keywords() (robot.libraries.BuiltIn.BuiltIn  
     method), 55  
 run\_process() (robot.libraries.Process.Process  
     method), 90  
 RunKeyword (class in robot.running.dynamicmethods),  
     513  
 RunKeywordRunner (class in  
     robot.running.librarykeywordrunner), 515
- ## S
- save() (robot.libdocpkg.model.LibraryDoc method),  
     38  
 save() (robot.parsing.model.blocks.File method), 306  
 save() (robot.result.executionresult.CombinedResult  
     method), 378  
 save() (robot.result.executionresult.Result method),  
     378  
 save\_xml() (robot.libraries.XML.XML method), 122  
 ScalarsAndListReturnValueResolver (class  
     in robot.variables.assigner), 568  
 ScalarsOnlyReturnValueResolver (class in  
     robot.variables.assigner), 568  
 ScalarVariableTableValue (class in  
     robot.variables.tablessetter), 573  
 Screenshot (class in robot.libraries.Screenshot), 95  
 ScreenshotTaker (class in  
     robot.libraries.Screenshot), 97  
 search() (robot.libdocpkg.consoleviewer.KeywordMatcher  
     method), 36  
 search() (robot.variables.search.VariableSearcher  
     method), 572  
 search\_variable() (in module  
     robot.variables.search), 572  
 Section (class in robot.parsing.model.blocks), 306  
 SectionHeader (class in  
     robot.parsing.model.statements), 315  
 SectionHeaderLexer (class in  
     robot.parsing.lexer.statementslexers), 294  
 SectionLexer (class in  
     robot.parsing.lexer.blocklexers), 284  
 SectionParser (class in  
     robot.parsing.parser.fileparser), 355  
 Sections (class in robot.parsing.lexer.sections), 290  
 sections\_class (robot.parsing.lexer.context.FileContext  
     attribute), 288  
 sections\_class (robot.parsing.lexer.context.InitFileContext  
     attribute), 289  
 sections\_class (robot.parsing.lexer.context.ResourceFileContext  
     attribute), 289  
 sections\_class (robot.parsing.lexer.context.TestCaseFileContext  
     attribute), 288  
 selection\_clear()  
     (robot.libraries.dialogs\_py.InputDialog  
     method), 144  
 selection\_clear()  
     (robot.libraries.dialogs\_py.MessageDialog  
     method), 130  
 selection\_clear()  
     (robot.libraries.dialogs\_py.MultipleSelectionDialog  
     method), 172  
 selection\_clear()  
     (robot.libraries.dialogs\_py.PassFailDialog  
     method), 186  
 selection\_clear()  
     (robot.libraries.dialogs\_py.SelectionDialog  
     method), 158  
 selection\_get() (robot.libraries.dialogs\_py.InputDialog  
     method), 144  
 selection\_get() (robot.libraries.dialogs\_py.MessageDialog  
     method), 130  
 selection\_get() (robot.libraries.dialogs\_py.MultipleSelectionDialog  
     method), 172  
 selection\_get() (robot.libraries.dialogs\_py.PassFailDialog  
     method), 186  
 selection\_get() (robot.libraries.dialogs\_py.SelectionDialog  
     method), 158  
 selection\_handle()  
     (robot.libraries.dialogs\_py.InputDialog  
     method), 144  
 selection\_handle()  
     (robot.libraries.dialogs\_py.MessageDialog  
     method), 130  
 selection\_handle()  
     (robot.libraries.dialogs\_py.MultipleSelectionDialog  
     method), 172  
 selection\_handle()  
     (robot.libraries.dialogs\_py.PassFailDialog  
     method), 186  
 selection\_handle()  
     (robot.libraries.dialogs\_py.SelectionDialog

- [method](#)), 158
- [selection\\_own\(\)](#) ([robot.libraries.dialogs\\_py.InputDialog](#) [method](#)), 144
- [selection\\_own\(\)](#) ([robot.libraries.dialogs\\_py.MessageDialog](#) [method](#)), 130
- [selection\\_own\(\)](#) ([robot.libraries.dialogs\\_py.MultipleSelectionDialog](#) [method](#)), 172
- [selection\\_own\(\)](#) ([robot.libraries.dialogs\\_py.PassFailDialog](#) [method](#)), 186
- [selection\\_own\(\)](#) ([robot.libraries.dialogs\\_py.SelectionDialog](#) [method](#)), 158
- [selection\\_own\\_get\(\)](#) ([robot.libraries.dialogs\\_py.InputDialog](#) [method](#)), 144
- [selection\\_own\\_get\(\)](#) ([robot.libraries.dialogs\\_py.MessageDialog](#) [method](#)), 130
- [selection\\_own\\_get\(\)](#) ([robot.libraries.dialogs\\_py.MultipleSelectionDialog](#) [method](#)), 172
- [selection\\_own\\_get\(\)](#) ([robot.libraries.dialogs\\_py.PassFailDialog](#) [method](#)), 186
- [selection\\_own\\_get\(\)](#) ([robot.libraries.dialogs\\_py.SelectionDialog](#) [method](#)), 158
- [SelectionDialog](#) (class in [robot.libraries.dialogs\\_py](#)), 150
- [send\(\)](#) ([robot.libraries.dialogs\\_py.InputDialog](#) [method](#)), 144
- [send\(\)](#) ([robot.libraries.dialogs\\_py.MessageDialog](#) [method](#)), 130
- [send\(\)](#) ([robot.libraries.dialogs\\_py.MultipleSelectionDialog](#) [method](#)), 172
- [send\(\)](#) ([robot.libraries.dialogs\\_py.PassFailDialog](#) [method](#)), 186
- [send\(\)](#) ([robot.libraries.dialogs\\_py.SelectionDialog](#) [method](#)), 158
- [send\\_content\(\)](#) ([robot.libraries.Remote.TimeoutHTTPSTransport](#) [method](#)), 95
- [send\\_content\(\)](#) ([robot.libraries.Remote.TimeoutHTTPSTransport](#) [method](#)), 95
- [send\\_headers\(\)](#) ([robot.libraries.Remote.TimeoutHTTPSTransport](#) [method](#)), 95
- [send\\_headers\(\)](#) ([robot.libraries.Remote.TimeoutHTTPSTransport](#) [method](#)), 95
- [send\\_request\(\)](#) ([robot.libraries.Remote.TimeoutHTTPSTransport](#) [method](#)), 95
- [send\\_request\(\)](#) ([robot.libraries.Remote.TimeoutHTTPSTransport](#) [method](#)), 84
- [send\\_signal\\_to\\_process\(\)](#) ([robot.libraries.Process.Process](#) [method](#)), 92
- [SEPARATOR](#) ([robot.parsing.lexer.tokens.END](#) attribute), 305
- [SEPARATOR](#) ([robot.parsing.lexer.tokens.EOS](#) attribute), 302
- [SEPARATOR](#) ([robot.parsing.lexer.tokens.Token](#) attribute), 301
- [SEPARATOR](#) ([robot.parsing.lexer.statemlexers.ForHeaderLexer](#) attribute), 296
- [seq2str\(\)](#) (in module [robot.utils.misc](#)), 565
- [seq2str2\(\)](#) (in module [robot.utils.misc](#)), 565
- [set\(\)](#) ([robot.result.keywordremover.RemovalMessage](#) [method](#)), 413
- [set\(\)](#) ([robot.variables.filesetter.VariableFileSetter](#) [method](#)), 569
- [set\(\)](#) ([robot.variables.tablesetter.VariableTableSetter](#) [method](#)), 573
- [set\\_debuglevel\(\)](#) ([robot.libraries.Telnet.TelnetConnection](#) [method](#)), 111
- [set\\_default\\_log\\_level\(\)](#) ([robot.libraries.Telnet.TelnetConnection](#) [method](#)), 107
- [set\\_earlier\\_failures\(\)](#) ([robot.errors.BreakLoop](#) [method](#)), 579
- [set\\_earlier\\_failures\(\)](#) ([robot.errors.ContinueLoop](#) [method](#)), 579
- [set\\_earlier\\_failures\(\)](#) ([robot.errors.ExecutionPassed](#) [method](#)), 578
- [set\\_earlier\\_failures\(\)](#) ([robot.errors.PassExecution](#) [method](#)), 578
- [set\\_earlier\\_failures\(\)](#) ([robot.errors.ReturnFromKeyword](#) [method](#)), 579
- [set\\_element\\_attribute\(\)](#) ([robot.libraries.XML.XML](#) [method](#)), 120
- [set\\_element\\_tag\(\)](#) ([robot.libraries.XML.XML](#) [method](#)), 119
- [set\\_element\\_text\(\)](#) ([robot.libraries.XML.XML](#) [method](#)), 119
- [set\\_elements\\_attribute\(\)](#) ([robot.libraries.XML.XML](#) [method](#)), 120
- [set\\_elements\\_tag\(\)](#) ([robot.libraries.XML.XML](#) [method](#)), 119
- [set\\_elements\\_text\(\)](#) ([robot.libraries.XML.XML](#) [method](#)), 120
- [set\\_encoding\(\)](#) ([robot.libraries.Telnet.TelnetConnection](#) [method](#)), 107
- [set\\_environment\\_variable\(\)](#) ([robot.libraries.OperatingSystem.OperatingSystem](#) [method](#)), 305
- [set\\_error\(\)](#) ([robot.parsing.lexer.tokens.END](#) [method](#)), 305
- [set\\_error\(\)](#) ([robot.parsing.lexer.tokens.EOS](#) [method](#)), 303
- [set\\_error\(\)](#) ([robot.parsing.lexer.tokens.Token](#) [method](#)), 303

- [method\), 301](#)
- [set\\_execution\\_mode\(\) \(robot.result.executionresult.CombinedResult method\), 378](#)
- [set\\_execution\\_mode\(\) \(robot.result.executionresult.Result method\), 378](#)
- [set\\_from\\_file\(\) \(robot.variables.scopes.GlobalVariables method\), 571](#)
- [set\\_from\\_file\(\) \(robot.variables.scopes.VariableScopes method\), 571](#)
- [set\\_from\\_file\(\) \(robot.variables.variables.Variables method\), 574](#)
- [set\\_from\\_variable\\_table\(\) \(robot.variables.scopes.GlobalVariables method\), 571](#)
- [set\\_from\\_variable\\_table\(\) \(robot.variables.scopes.VariableScopes method\), 571](#)
- [set\\_from\\_variable\\_table\(\) \(robot.variables.variables.Variables method\), 574](#)
- [set\\_global\(\) \(robot.variables.scopes.SetVariables method\), 572](#)
- [set\\_global\(\) \(robot.variables.scopes.VariableScopes method\), 571](#)
- [set\\_global\\_variable\(\) \(robot.libraries.BuiltIn.BuiltIn method\), 55](#)
- [set\\_if\\_removed\(\) \(robot.result.keywordremover.RemoveMessage method\), 413](#)
- [set\\_keyword\(\) \(robot.variables.scopes.SetVariables method\), 572](#)
- [set\\_keyword\(\) \(robot.variables.scopes.VariableScopes method\), 571](#)
- [set\\_keyword\\_timeout\(\) \(robot.running.timeouts.TestTimeout method\), 510](#)
- [set\\_level\(\) \(in module robot.output.pyloggingconf\), 277](#)
- [set\\_level\(\) \(robot.output.filelogger.FileLogger method\), 270](#)
- [set\\_level\(\) \(robot.output.logger.Logger method\), 274](#)
- [set\\_level\(\) \(robot.output.loggerhelper.AbstractLogger method\), 274](#)
- [set\\_level\(\) \(robot.output.loggerhelper.IsLogged method\), 276](#)
- [set\\_level\(\) \(robot.output.output.Output method\), 276](#)
- [set\\_library\\_search\\_order\(\) \(robot.libraries.BuiltIn.BuiltIn method\), 55](#)
- [set\\_list\\_value\(\) \(robot.libraries.Collections.Collections method\), 71](#)
- [set\\_local\\_variable\(\) \(robot.libraries.BuiltIn.BuiltIn method\), 55](#)
- [set\\_local\\_variable\(\) \(robot.variables.scopes.VariableScopes method\), 571](#)
- [set\\_log\\_level\(\) \(robot.libraries.BuiltIn.BuiltIn method\), 56](#)
- [set\\_log\\_level\(\) \(robot.output.listeners.LibraryListeners method\), 272](#)
- [set\\_log\\_level\(\) \(robot.output.listeners.Listeners method\), 272](#)
- [set\\_log\\_level\(\) \(robot.output.output.Output method\), 276](#)
- [set\\_log\\_level\(\) \(robot.output.xmllogger.XmlLogger method\), 278](#)
- [set\\_log\\_level\(\) \(robot.reporting.outputwriter.OutputWriter method\), 361](#)
- [set\\_modified\\_time\(\) \(robot.libraries.OperatingSystem.OperatingSystem method\), 86](#)
- [set\\_name\(\) \(robot.output.pyloggingconf.RobotHandler method\), 278](#)
- [set\\_newline\(\) \(robot.libraries.Telnet.TelnetConnection method\), 107](#)
- [set\\_option\\_negotiation\\_callback\(\) \(robot.libraries.Telnet.TelnetConnection method\), 111](#)
- [set\\_option\\_negotiation\\_callback\(\) \(robot.libraries.Telnet.TelnetConnection method\), 107](#)
- [set\\_screenshot\\_directory\(\) \(robot.libraries.Screenshot.Screenshot method\), 96](#)
- [set\\_search\\_order\(\) \(robot.running.namespace.Namespace method\), 540](#)
- [set\\_suite\(\) \(robot.variables.scopes.SetVariables method\), 572](#)
- [set\\_suite\(\) \(robot.variables.scopes.VariableScopes method\), 571](#)
- [set\\_suite\\_documentation\(\) \(robot.libraries.BuiltIn.BuiltIn method\), 56](#)
- [set\\_suite\\_metadata\(\) \(robot.libraries.BuiltIn.BuiltIn method\), 56](#)
- [set\\_suite\\_variable\(\) \(robot.libraries.BuiltIn.BuiltIn method\), 56](#)
- [set\\_tags\(\) \(robot.libraries.BuiltIn.BuiltIn method\), 57](#)
- [set\\_tags\(\) \(robot.model.testsuite.TestSuite method\), 251](#)



`set_tags()` (*robot.result.model.TestSuite* method), 462

`set_tags()` (*robot.running.model.TestSuite* method), 537

`set_task_variable()` (*robot.libraries.BuiltIn.BuiltIn* method), 57

`set_telnetlib_log_level()` (*robot.libraries.Telnet.TelnetConnection* method), 107

`set_test()` (*robot.variables.scopes.SetVariables* method), 572

`set_test()` (*robot.variables.scopes.VariableScopes* method), 571

`set_test_documentation()` (*robot.libraries.BuiltIn.BuiltIn* method), 57

`set_test_message()` (*robot.libraries.BuiltIn.BuiltIn* method), 57

`set_test_variable()` (*robot.libraries.BuiltIn.BuiltIn* method), 57

`set_timeout()` (*robot.libraries.Telnet.TelnetConnection* method), 107

`set_to_dictionary()` (*robot.libraries.Collections.Collections* method), 71

`set_variable()` (*robot.libraries.BuiltIn.BuiltIn* method), 58

`set_variable_if()` (*robot.libraries.BuiltIn.BuiltIn* method), 58

`SetConverter` (class in *robot.running.arguments.typeconverters*), 501

`setdefault()` (*robot.model.metadata.Metadata* method), 230

`setdefault()` (*robot.utils.dotdict.DotDict* method), 559

`setdefault()` (*robot.utils.normalizing.NormalizedDict* method), 566

`setFormatter()` (*robot.output.pyloggingconf.RobotHandler* method), 278

`setLevel()` (*robot.output.pyloggingconf.RobotHandler* method), 278

`setting()` (*robot.parsing.lexer.sections.InitFileSections* method), 292

`setting()` (*robot.parsing.lexer.sections.ResourceFileSections* method), 291

`setting()` (*robot.parsing.lexer.sections.Sections* method), 291

`setting()` (*robot.parsing.lexer.sections.TestCaseFileSections* method), 291

`SETTING_HEADER` (*robot.parsing.lexer.tokens.END* attribute), 305

`SETTING_HEADER` (*robot.parsing.lexer.tokens.EOS* attribute), 302

`SETTING_HEADER` (*robot.parsing.lexer.tokens.Token* attribute), 299

`setting_markers` (*robot.parsing.lexer.sections.InitFileSections* attribute), 292

`setting_markers` (*robot.parsing.lexer.sections.ResourceFileSections* attribute), 291

`setting_markers` (*robot.parsing.lexer.sections.Sections* attribute), 290

`setting_markers` (*robot.parsing.lexer.sections.TestCaseFileSections* attribute), 291

`setting_section()` (*robot.parsing.lexer.context.FileContext* method), 288

`setting_section()` (*robot.parsing.lexer.context.InitFileContext* method), 289

`setting_section()` (*robot.parsing.lexer.context.ResourceFileContext* method), 289

`setting_section()` (*robot.parsing.lexer.context.TestCaseFileContext* method), 289

`SETTING_TOKENS` (*robot.parsing.lexer.tokens.END* attribute), 305

`SETTING_TOKENS` (*robot.parsing.lexer.tokens.EOS* attribute), 302

`SETTING_TOKENS` (*robot.parsing.lexer.tokens.Token* attribute), 301

`SettingLexer` (class in *robot.parsing.lexer.statementlexers*), 296

`Settings` (class in *robot.parsing.lexer.settings*), 292

`settings_class` (*robot.parsing.lexer.context.FileContext* attribute), 288

`settings_class` (*robot.parsing.lexer.context.InitFileContext* attribute), 289

`settings_class` (*robot.parsing.lexer.context.KeywordContext* attribute), 290

`settings_class` (*robot.parsing.lexer.context.LexingContext* attribute), 288

`settings_class` (*robot.parsing.lexer.context.ResourceFileContext* attribute), 289

`settings_class` (*robot.parsing.lexer.context.TestCaseContext* attribute), 289

`settings_class` (*robot.parsing.lexer.context.TestCaseFileContext* attribute), 288

`SettingsBuilder` (class in *robot.running.builder.transformers*), 506

`SettingSection` (class in *robot.parsing.model.blocks*), 307

`SettingSectionHeaderLexer` (class in *robot.parsing.lexer.statementlexers*), 294

- SettingSectionLexer (class robot.parsing.lexer.blocklexers), 284
- SettingSectionParser (class robot.parsing.parser.fileparser), 355
- Setup (class in robot.parsing.model.statements), 329
- SETUP (robot.model.body.BodyItem attribute), 193
- SETUP (robot.model.control.Break attribute), 214
- SETUP (robot.model.control.Continue attribute), 213
- SETUP (robot.model.control.For attribute), 204
- SETUP (robot.model.control.If attribute), 208
- SETUP (robot.model.control.IfBranch attribute), 207
- SETUP (robot.model.control.Return attribute), 212
- SETUP (robot.model.control.Try attribute), 211
- SETUP (robot.model.control.TryBranch attribute), 209
- SETUP (robot.model.control.While attribute), 205
- SETUP (robot.model.keyword.Keyword attribute), 225
- setup (robot.model.keyword.Keywords attribute), 226
- SETUP (robot.model.message.Message attribute), 228
- setup (robot.model.testcase.TestCase attribute), 248
- setup (robot.model.testsuite.TestSuite attribute), 250
- SETUP (robot.output.loggerhelper.Message attribute), 275
- SETUP (robot.parsing.lexer.tokens.END attribute), 305
- SETUP (robot.parsing.lexer.tokens.EOS attribute), 302
- SETUP (robot.parsing.lexer.tokens.Token attribute), 300
- SETUP (robot.result.model.Break attribute), 452
- SETUP (robot.result.model.Continue attribute), 450
- SETUP (robot.result.model.For attribute), 431
- SETUP (robot.result.model.ForIteration attribute), 429
- SETUP (robot.result.model.If attribute), 441
- SETUP (robot.result.model.IfBranch attribute), 438
- SETUP (robot.result.model.Keyword attribute), 455
- SETUP (robot.result.model.Message attribute), 427
- SETUP (robot.result.model.Return attribute), 448
- setup (robot.result.model.TestCase attribute), 459
- setup (robot.result.model.TestSuite attribute), 462
- SETUP (robot.result.model.Try attribute), 445
- SETUP (robot.result.model.TryBranch attribute), 443
- SETUP (robot.result.model.While attribute), 436
- SETUP (robot.result.model.WhileIteration attribute), 434
- setup (robot.running.builder.testsettings.TestDefaults attribute), 505
- setup (robot.running.builder.testsettings.TestSettings attribute), 506
- SETUP (robot.running.model.Break attribute), 531
- SETUP (robot.running.model.Continue attribute), 530
- SETUP (robot.running.model.For attribute), 520
- SETUP (robot.running.model.If attribute), 525
- SETUP (robot.running.model.IfBranch attribute), 523
- SETUP (robot.running.model.Keyword attribute), 518
- SETUP (robot.running.model.Return attribute), 529
- setup (robot.running.model.TestCase attribute), 534
- setup (robot.running.model.TestSuite attribute), 537
- SETUP (robot.running.model.Try attribute), 527
- in SETUP (robot.running.model.TryBranch attribute), 526
- SETUP (robot.running.model.While attribute), 522
- in setup\_executed() (robot.running.status.SuiteStatus method), 545
- setup\_executed() (robot.running.status.TestStatus method), 546
- setup\_message (robot.running.status.ParentMessage attribute), 546
- setup\_message (robot.running.status.SuiteMessage attribute), 546
- setup\_message (robot.running.status.TestMessage attribute), 546
- setup\_skipped\_message (robot.running.status.ParentMessage attribute), 546
- setup\_skipped\_message (robot.running.status.SuiteMessage attribute), 546
- setup\_skipped\_message (robot.running.status.TestMessage attribute), 546
- setvar() (robot.libraries.dialogs\_py.InputDialog method), 144
- setvar() (robot.libraries.dialogs\_py.MessageDialog method), 130
- setvar() (robot.libraries.dialogs\_py.MultipleSelectionDialog method), 172
- setvar() (robot.libraries.dialogs\_py.PassFailDialog method), 186
- setvar() (robot.libraries.dialogs\_py.SelectionDialog method), 158
- SetVariables (class in robot.variables.scopes), 571
- shortdoc (robot.libdocpkg.model.KeywordDoc attribute), 38
- shortdoc (robot.running.usererrorhandler.UserErrorHandler attribute), 551
- shortdoc (robot.running.userkeyword.EmbeddedArgumentsHandler attribute), 552
- shortdoc (robot.running.userkeyword.UserKeywordHandler attribute), 552
- should\_be\_byte\_string() (robot.libraries.String.String method), 102
- should\_be\_empty() (robot.libraries.BuiltIn.BuiltIn method), 58
- should\_be\_equal() (robot.libraries.BuiltIn.BuiltIn method), 58
- should\_be\_equal\_as\_integers() (robot.libraries.BuiltIn.BuiltIn method), 59
- should\_be\_equal\_as\_numbers() (robot.libraries.BuiltIn.BuiltIn method), 59
- should\_be\_equal\_as\_strings() (robot.libraries.BuiltIn.BuiltIn method),

59  
`should_be_lower_case()`  
    (*robot.libraries.String.String* method), 102  
`should_be_string()` (*robot.libraries.String.String*  
    method), 101  
`should_be_title_case()`  
    (*robot.libraries.String.String* method), 102  
`should_be_true()` (*robot.libraries.BuiltIn.BuiltIn*  
    method), 59  
`should_be_unicode_string()`  
    (*robot.libraries.String.String* method), 102  
`should_be_upper_case()`  
    (*robot.libraries.String.String* method), 102  
`should_contain()` (*robot.libraries.BuiltIn.BuiltIn*  
    method), 60  
`should_contain_any()`  
    (*robot.libraries.BuiltIn.BuiltIn* method),  
    60  
`should_contain_match()`  
    (*robot.libraries.Collections.Collections*  
    method), 66  
`should_contain_x_times()`  
    (*robot.libraries.BuiltIn.BuiltIn* method),  
    60  
`should_end_with()` (*robot.libraries.BuiltIn.BuiltIn*  
    method), 61  
`should_exist()` (*robot.libraries.OperatingSystem.OperatingSystem*  
    method), 80  
`should_match()` (*robot.libraries.BuiltIn.BuiltIn*  
    method), 61  
`should_match_regexp()`  
    (*robot.libraries.BuiltIn.BuiltIn* method),  
    61  
`should_not_be_empty()`  
    (*robot.libraries.BuiltIn.BuiltIn* method),  
    61  
`should_not_be_equal()`  
    (*robot.libraries.BuiltIn.BuiltIn* method),  
    61  
`should_not_be_equal_as_integers()`  
    (*robot.libraries.BuiltIn.BuiltIn* method),  
    62  
`should_not_be_equal_as_numbers()`  
    (*robot.libraries.BuiltIn.BuiltIn* method),  
    62  
`should_not_be_equal_as_strings()`  
    (*robot.libraries.BuiltIn.BuiltIn* method),  
    62  
`should_not_be_string()`  
    (*robot.libraries.String.String* method), 102  
`should_not_be_true()`  
    (*robot.libraries.BuiltIn.BuiltIn* method),  
    62  
`should_not_contain()`  
    (*robot.libraries.BuiltIn.BuiltIn* method),  
    62  
`should_not_contain_any()`  
    (*robot.libraries.BuiltIn.BuiltIn* method),  
    63  
`should_not_contain_match()`  
    (*robot.libraries.Collections.Collections*  
    method), 67  
`should_not_end_with()`  
    (*robot.libraries.BuiltIn.BuiltIn* method),  
    63  
`should_not_exist()`  
    (*robot.libraries.OperatingSystem.OperatingSystem*  
    method), 81  
`should_not_match()`  
    (*robot.libraries.BuiltIn.BuiltIn* method),  
    63  
`should_not_match_regexp()`  
    (*robot.libraries.BuiltIn.BuiltIn* method),  
    63  
`should_not_start_with()`  
    (*robot.libraries.BuiltIn.BuiltIn* method),  
    63  
`should_start_with()`  
    (*robot.libraries.BuiltIn.BuiltIn* method),  
    63  
`show()` (*robot.libdocpkg.consoleviewer.ConsoleViewer*  
    method), 36  
`show()` (*robot.libraries.dialogs\_py.InputDialog*  
    method), 144  
`show()` (*robot.libraries.dialogs\_py.MessageDialog*  
    method), 130  
`show()` (*robot.libraries.dialogs\_py.MultipleSelectionDialog*  
    method), 172  
`show()` (*robot.libraries.dialogs\_py.PassFailDialog*  
    method), 186  
`show()` (*robot.libraries.dialogs\_py.SelectionDialog*  
    method), 158  
`single_request()` (*robot.libraries.Remote.TimeoutHTTPSTransport*  
    method), 95  
`single_request()` (*robot.libraries.Remote.TimeoutHTTPSTransport*  
    method), 95  
`single_value` (*robot.parsing.lexer.settings.InitFileSettings*  
    attribute), 292  
`single_value` (*robot.parsing.lexer.settings.KeywordSettings*  
    attribute), 293  
`single_value` (*robot.parsing.lexer.settings.ResourceFileSettings*  
    attribute), 293  
`single_value` (*robot.parsing.lexer.settings.Settings*  
    attribute), 292  
`single_value` (*robot.parsing.lexer.settings.TestCaseFileSettings*  
    attribute), 292  
`single_value` (*robot.parsing.lexer.settings.TestCaseSettings*  
    attribute), 293



SingleTagPattern (class in *robot.model.tags*), 242  
 SingleType (class in *robot.parsing.lexer.statemlexers*), 294  
 SingleValue (class in *robot.parsing.model.statements*), 312  
 size() (*robot.libraries.dialogs\_py.InputDialog* method), 144  
 size() (*robot.libraries.dialogs\_py.MessageDialog* method), 130  
 size() (*robot.libraries.dialogs\_py.MultipleSelectionDialog* method), 172  
 size() (*robot.libraries.dialogs\_py.PassFailDialog* method), 186  
 size() (*robot.libraries.dialogs\_py.SelectionDialog* method), 158  
 sizefrom() (*robot.libraries.dialogs\_py.InputDialog* method), 144  
 sizefrom() (*robot.libraries.dialogs\_py.MessageDialog* method), 130  
 sizefrom() (*robot.libraries.dialogs\_py.MultipleSelectionDialog* method), 172  
 sizefrom() (*robot.libraries.dialogs\_py.PassFailDialog* method), 186  
 sizefrom() (*robot.libraries.dialogs\_py.SelectionDialog* method), 158  
 skip (*robot.conf.settings.RobotSettings* attribute), 32  
 SKIP (*robot.result.model.Break* attribute), 452  
 SKIP (*robot.result.model.Continue* attribute), 450  
 SKIP (*robot.result.model.For* attribute), 432  
 SKIP (*robot.result.model.ForIteration* attribute), 429  
 SKIP (*robot.result.model.If* attribute), 441  
 SKIP (*robot.result.model.IfBranch* attribute), 438  
 SKIP (*robot.result.model.Keyword* attribute), 455  
 SKIP (*robot.result.model.Return* attribute), 448  
 SKIP (*robot.result.model.StatusMixin* attribute), 428  
 SKIP (*robot.result.model.TestCase* attribute), 458  
 SKIP (*robot.result.model.TestSuite* attribute), 460  
 SKIP (*robot.result.model.Try* attribute), 445  
 SKIP (*robot.result.model.TryBranch* attribute), 443  
 SKIP (*robot.result.model.While* attribute), 436  
 SKIP (*robot.result.model.WhileIteration* attribute), 434  
 skip() (*robot.libraries.BuiltIn.BuiltIn* method), 63  
 skip() (*robot.output.filelogger.FileLogger* method), 270  
 skip() (*robot.output.logger.Logger* method), 274  
 skip() (*robot.output.loggerhelper.AbstractLogger* method), 274  
 skip() (*robot.output.output.Output* method), 276  
 skip\_if() (*robot.libraries.BuiltIn.BuiltIn* method), 63  
 skip\_on\_failure (*robot.conf.settings.RobotSettings* attribute), 32  
 skip\_on\_failure\_after\_tag\_changes (*robot.running.status.TestStatus* attribute), 545  
 skip\_teardown\_on\_exit (*robot.conf.settings.RobotSettings* attribute), 32  
 SkipExecution, 13  
 skipped (*robot.model.stats.Stat* attribute), 240  
 skipped (*robot.model.totalstatistics.TotalStatistics* attribute), 253  
 skipped (*robot.result.model.Break* attribute), 453  
 skipped (*robot.result.model.Continue* attribute), 451  
 skipped (*robot.result.model.For* attribute), 433  
 skipped (*robot.result.model.ForIteration* attribute), 430  
 skipped (*robot.result.model.If* attribute), 442  
 skipped (*robot.result.model.IfBranch* attribute), 440  
 skipped (*robot.result.model.Keyword* attribute), 456  
 skipped (*robot.result.model.Return* attribute), 449  
 skipped (*robot.result.model.StatusMixin* attribute), 428  
 skipped (*robot.result.model.TestCase* attribute), 459  
 skipped (*robot.result.model.TestSuite* attribute), 460  
 skipped (*robot.result.model.Try* attribute), 447  
 skipped (*robot.result.model.TryBranch* attribute), 444  
 skipped (*robot.result.model.While* attribute), 437  
 skipped (*robot.result.model.WhileIteration* attribute), 435  
 skipped\_tags (*robot.conf.settings.RobotSettings* attribute), 32  
 slaves() (*robot.libraries.dialogs\_py.InputDialog* method), 144  
 slaves() (*robot.libraries.dialogs\_py.MessageDialog* method), 130  
 slaves() (*robot.libraries.dialogs\_py.MultipleSelectionDialog* method), 172  
 slaves() (*robot.libraries.dialogs\_py.PassFailDialog* method), 186  
 slaves() (*robot.libraries.dialogs\_py.SelectionDialog* method), 158  
 sleep() (*robot.libraries.BuiltIn.BuiltIn* method), 64  
 sock\_avail() (*robot.libraries.Telnet.TelnetConnection* method), 111  
 sort() (*robot.model.body.BaseBody* method), 195  
 sort() (*robot.model.body.Body* method), 197  
 sort() (*robot.model.body.Branches* method), 199  
 sort() (*robot.model.itemlist.ItemList* method), 224  
 sort() (*robot.model.keyword.Keywords* method), 227  
 sort() (*robot.model.message.Messages* method), 229  
 sort() (*robot.model.testcase.TestCases* method), 250  
 sort() (*robot.model.testsuite.TestSuites* method), 253  
 sort() (*robot.result.model.Body* method), 423  
 sort() (*robot.result.model.Branches* method), 425  
 sort() (*robot.result.model.Iterations* method), 426  
 sort() (*robot.running.model.Body* method), 517  
 sort() (*robot.running.model.Imports* method), 539  
 sort\_list() (*robot.libraries.Collections.Collections* method), 72  
 source (*robot.model.testcase.TestCase* attribute), 249

- source (*robot.model.testsuite.TestSuite* attribute), 250
- source (*robot.result.executionresult.Result* attribute), 377
- source (*robot.result.model.TestCase* attribute), 459
- source (*robot.result.model.TestSuite* attribute), 462
- source (*robot.running.model.Break* attribute), 531
- source (*robot.running.model.Continue* attribute), 530
- source (*robot.running.model.For* attribute), 520
- source (*robot.running.model.If* attribute), 524
- source (*robot.running.model.IfBranch* attribute), 523
- source (*robot.running.model.Keyword* attribute), 518
- source (*robot.running.model.Return* attribute), 528
- source (*robot.running.model.TestCase* attribute), 532
- source (*robot.running.model.TestSuite* attribute), 538
- source (*robot.running.model.Try* attribute), 527
- source (*robot.running.model.TryBranch* attribute), 526
- source (*robot.running.model.UserKeyword* attribute), 539
- source (*robot.running.model.While* attribute), 521
- sourcename (*robot.result.model.Keyword* attribute), 454
- split\_command\_line() (*robot.libraries.Process.Process* method), 93
- split\_extension() (*robot.libraries.OperatingSystem.OperatingSystem* method), 86
- split\_from\_equals() (in module *robot.utils.escaping*), 560
- split\_log (*robot.conf.settings.RebotSettings* attribute), 33
- split\_log (*robot.conf.settings.RobotSettings* attribute), 32
- split\_path() (*robot.libraries.OperatingSystem.OperatingSystem* method), 85
- split\_string() (*robot.libraries.String.String* method), 101
- split\_string\_from\_right() (*robot.libraries.String.String* method), 101
- split\_string\_to\_characters() (*robot.libraries.String.String* method), 101
- split\_to\_lines() (*robot.libraries.String.String* method), 99
- SplitLogWriter (class in *robot.reporting.jswriter*), 359
- STANDARD (*robot.libdocpkg.datatypes.TypeDoc* attribute), 36
- start() (*robot.result.xmlelementhandlers.ArgumentHandler* method), 490
- start() (*robot.result.xmlelementhandlers.ArgumentsHandler* method), 489
- start() (*robot.result.xmlelementhandlers.AssignHandler* method), 489
- start() (*robot.result.xmlelementhandlers.BranchHandler* method), 485
- start() (*robot.result.xmlelementhandlers.BreakHandler* method), 486
- start() (*robot.result.xmlelementhandlers.ContinueHandler* method), 486
- start() (*robot.result.xmlelementhandlers.DocHandler* method), 487
- start() (*robot.result.xmlelementhandlers.ElementHandler* method), 483
- start() (*robot.result.xmlelementhandlers.ErrorMessageHandler* method), 490
- start() (*robot.result.xmlelementhandlers.ErrorsHandler* method), 490
- start() (*robot.result.xmlelementhandlers.ForHandler* method), 484
- start() (*robot.result.xmlelementhandlers.IfHandler* method), 485
- start() (*robot.result.xmlelementhandlers.IterationHandler* method), 485
- start() (*robot.result.xmlelementhandlers.KeywordHandler* method), 484
- start() (*robot.result.xmlelementhandlers.MessageHandler* method), 487
- start() (*robot.result.xmlelementhandlers.MetadataHandler* method), 487
- start() (*robot.result.xmlelementhandlers.MetadataItemHandler* method), 488
- start() (*robot.result.xmlelementhandlers.MetaHandler* method), 488
- start() (*robot.result.xmlelementhandlers.PatternHandler* method), 486
- start() (*robot.result.xmlelementhandlers.ReturnHandler* method), 486
- start() (*robot.result.xmlelementhandlers.RobotHandler* method), 483
- start() (*robot.result.xmlelementhandlers.RootHandler* method), 483
- start() (*robot.result.xmlelementhandlers.StatisticsHandler* method), 491
- start() (*robot.result.xmlelementhandlers.StatusHandler* method), 487
- start() (*robot.result.xmlelementhandlers.SuiteHandler* method), 483
- start() (*robot.result.xmlelementhandlers.TagHandler* method), 488
- start() (*robot.result.xmlelementhandlers.TagsHandler* method), 488
- start() (*robot.result.xmlelementhandlers.TestHandler* method), 484
- start() (*robot.result.xmlelementhandlers.TimeoutHandler* method), 489
- start() (*robot.result.xmlelementhandlers.TryHandler* method), 485
- start() (*robot.result.xmlelementhandlers.ValueHandler* method), 485

*method*), 490  
*start()* (*robot.result.xmlelementhandlers.VarHandler*  
*method*), 489  
*start()* (*robot.result.xmlelementhandlers.WhileHandler*  
*method*), 484  
*start()* (*robot.result.xmlelementhandlers.XmlElementHandler*  
*method*), 483  
*start()* (*robot.running.timeouts.KeywordTimeout*  
*method*), 510  
*start()* (*robot.running.timeouts.TestTimeout* *method*),  
510  
*start()* (*robot.utils.markupwriters.HtmlWriter*  
*method*), 564  
*start()* (*robot.utils.markupwriters.NullMarkupWriter*  
*method*), 565  
*start()* (*robot.utils.markupwriters.XmlWriter*  
*method*), 565  
*start\_block()* (*robot.parsing.model.blocks.ValidationContext*  
*method*), 310  
*start\_body\_item()*  
(*robot.conf.gatherfailed.GatherFailedSuites*  
*method*), 28  
*start\_body\_item()*  
(*robot.conf.gatherfailed.GatherFailedTests*  
*method*), 24  
*start\_body\_item()*  
(*robot.model.configurer.SuiteConfigurer*  
*method*), 200  
*start\_body\_item()*  
(*robot.model.filter.EmptySuiteRemover*  
*method*), 216  
*start\_body\_item()* (*robot.model.filter.Filter*  
*method*), 221  
*start\_body\_item()*  
(*robot.model.modifier.ModelModifier* *method*),  
232  
*start\_body\_item()*  
(*robot.model.statistics.StatisticsBuilder*  
*method*), 237  
*start\_body\_item()*  
(*robot.model.tagsetter.TagSetter* *method*),  
244  
*start\_body\_item()*  
(*robot.model.totalstatistics.TotalStatisticsBuilder*  
*method*), 255  
*start\_body\_item()*  
(*robot.model.visitor.SuiteVisitor* *method*),  
263  
*start\_body\_item()*  
(*robot.output.console.dotted.StatusReporter*  
*method*), 265  
*start\_body\_item()*  
(*robot.output.xmllogger.XmlLogger* *method*),  
281  
*start\_body\_item()*  
(*robot.reporting.outputwriter.OutputWriter*  
*method*), 361  
*start\_body\_item()*  
(*robot.reporting.xunitwriter.XUnitFileWriter*  
*method*), 368  
*start\_body\_item()*  
(*robot.result.configurer.SuiteConfigurer*  
*method*), 373  
*start\_body\_item()*  
(*robot.result.keywordremover.AllKeywordsRemover*  
*method*), 381  
*start\_body\_item()*  
(*robot.result.keywordremover.ByNameKeywordRemover*  
*method*), 389  
*start\_body\_item()*  
(*robot.result.keywordremover.ByTagKeywordRemover*  
*method*), 393  
*start\_body\_item()*  
(*robot.result.keywordremover.ForLoopItemsRemover*  
*method*), 397  
*start\_body\_item()*  
(*robot.result.keywordremover.PassedKeywordRemover*  
*method*), 385  
*start\_body\_item()*  
(*robot.result.keywordremover.WaitUntilKeywordSucceedsRemover*  
*method*), 406  
*start\_body\_item()*  
(*robot.result.keywordremover.WarningAndErrorFinder*  
*method*), 410  
*start\_body\_item()*  
(*robot.result.keywordremover.WhileLoopItemsRemover*  
*method*), 401  
*start\_body\_item()* (*robot.result.merger.Merger*  
*method*), 414  
*start\_body\_item()*  
(*robot.result.messagefilter.MessageFilter*  
*method*), 419  
*start\_body\_item()*  
(*robot.result.resultbuilder.RemoveKeywords*  
*method*), 466  
*start\_body\_item()*  
(*robot.result.suiteteardownfailed.SuiteTeardownFailed*  
*method*), 475  
*start\_body\_item()*  
(*robot.result.suiteteardownfailed.SuiteTeardownFailureHandler*  
*method*), 470  
*start\_body\_item()*  
(*robot.result.visitor.ResultVisitor* *method*),  
479  
*start\_body\_item()*  
(*robot.running.randomizer.Randomizer*  
*method*), 542  
*start\_body\_item()*

(robot.running.suiterunner.SuiteRunner  
 method), 548

start\_break() (robot.conf.gatherfailed.GatherFailedSuites  
 method), 28

start\_break() (robot.conf.gatherfailed.GatherFailedTests  
 method), 24

start\_break() (robot.model.configurer.SuiteConfigurer  
 method), 201

start\_break() (robot.model.filter.EmptySuiteRemover  
 method), 217

start\_break() (robot.model.filter.Filter  
 method), 221

start\_break() (robot.model.modifier.ModelModifier  
 method), 232

start\_break() (robot.model.statistics.StatisticsBuilder  
 method), 237

start\_break() (robot.model.tagsetter.TagSetter  
 method), 244

start\_break() (robot.model.totalstatistics.TotalStatisticsBuilder  
 method), 255

start\_break() (robot.model.visitor.SuiteVisitor  
 method), 262

start\_break() (robot.output.console.dotted.StatusReporter  
 method), 265

start\_break() (robot.output.xmllogger.XmlLogger  
 method), 280

start\_break() (robot.reporting.outputwriter.OutputWriter  
 method), 361

start\_break() (robot.reporting.xunitwriter.XUnitFileWriter  
 method), 368

start\_break() (robot.result.configurer.SuiteConfigurer  
 method), 374

start\_break() (robot.result.keywordremover.AllKeywordsRemover  
 method), 381

start\_break() (robot.result.keywordremover.ByNameKeywordRemover  
 method), 389

start\_break() (robot.result.keywordremover.ByTagKeywordRemover  
 method), 393

start\_break() (robot.result.keywordremover.ForLoopItemsRemover  
 method), 397

start\_break() (robot.result.keywordremover.PassedKeywordRemover  
 method), 385

start\_break() (robot.result.keywordremover.WaitUntilKeywordSucceedsRemover  
 method), 406

start\_break() (robot.result.keywordremover.WarningAndErrorFinder  
 method), 410

start\_break() (robot.result.keywordremover.WhileLoopItemsRemover  
 method), 402

start\_break() (robot.result.merger.Merger  
 method), 414

start\_break() (robot.result.messagefilter.MessageFilter  
 method), 419

start\_break() (robot.result.resultbuilder.RemoveKeywords  
 method), 466

start\_break() (robot.result.suitetardownfailed.SuiteTeardownFailed  
 method), 475

start\_break() (robot.result.suitetardownfailed.SuiteTeardownFailure  
 method), 470

start\_break() (robot.result.visitor.ResultVisitor  
 method), 480

start\_break() (robot.running.randomizer.Randomizer  
 method), 542

start\_break() (robot.running.suiterunner.SuiteRunner  
 method), 548

start\_continue() (robot.conf.gatherfailed.GatherFailedSuites  
 method), 29

start\_continue() (robot.conf.gatherfailed.GatherFailedTests  
 method), 24

start\_continue() (robot.model.configurer.SuiteConfigurer  
 method), 201

start\_continue() (robot.model.filter.EmptySuiteRemover  
 method), 217

start\_continue() (robot.model.filter.Filter  
 method), 221

start\_continue() (robot.model.modifier.ModelModifier  
 method), 232

start\_continue() (robot.model.statistics.StatisticsBuilder  
 method), 237

start\_continue() (robot.model.tagsetter.TagSetter  
 method), 244

start\_continue() (robot.model.totalstatistics.TotalStatisticsBuilder  
 method), 255

start\_continue() (robot.model.visitor.SuiteVisitor  
 method), 262

start\_continue() (robot.output.console.dotted.StatusReporter  
 method), 265

start\_continue() (robot.output.xmllogger.XmlLogger  
 method), 280

start\_continue() (robot.reporting.outputwriter.OutputWriter  
 method), 361

start\_continue() (robot.reporting.xunitwriter.XUnitFileWriter  
 method), 368

start\_continue() (robot.result.configurer.SuiteConfigurer  
 method), 374

start\_continue() (robot.result.keywordremover.AllKeywordsRemover  
 method), 381

start\_continue() (robot.result.keywordremover.ByNameKeywordRemover  
 method), 389

start\_continue() (robot.result.keywordremover.ByTagKeywordRemover  
 method), 393

start\_continue() (robot.result.keywordremover.ForLoopItemsRemover  
 method), 397

start\_continue() (robot.result.keywordremover.PassedKeywordRemover  
 method), 385

start\_continue() (robot.result.keywordremover.WaitUntilKeywordSucceedsRemover  
 method), 406

start\_continue() (robot.result.keywordremover.WarningAndErrorFinder  
 method), 410

start\_continue() (robot.result.keywordremover.WhileLoopItemsRemover  
 method), 402

start\_continue() (robot.result.merger.Merger  
 method), 414

start\_continue() (robot.result.messagefilter.MessageFilter  
 method), 419

start\_continue() (robot.result.resultbuilder.RemoveKeywords  
 method), 466



[start\\_continue\(\) \(robot.result.keywordremover.WhileLoopItemsRemover method\), 279](#)  
[start\\_continue\(\) \(robot.result.merger.Merger method\), 402](#)  
[start\\_continue\(\) \(robot.result.messagefilter.MessageFilter method\), 415](#)  
[start\\_continue\(\) \(robot.result.messagefilter.MessageFilter method\), 419](#)  
[start\\_continue\(\) \(robot.result.resultbuilder.RemoveKeywords method\), 374](#)  
[start\\_continue\(\) \(robot.result.resultbuilder.RemoveKeywords method\), 466](#)  
[start\\_continue\(\) \(robot.result.suitetardownfailed.SuiteTeardownFailed method\), 381](#)  
[start\\_continue\(\) \(robot.result.suitetardownfailed.SuiteTeardownFailed method\), 475](#)  
[start\\_continue\(\) \(robot.result.suitetardownfailed.SuiteTeardownFailureHandler method\), 471](#)  
[start\\_continue\(\) \(robot.result.visitor.ResultVisitor method\), 480](#)  
[start\\_continue\(\) \(robot.running.randomizer.Randomizer method\), 542](#)  
[start\\_continue\(\) \(robot.running.suiterunner.SuiteRunner method\), 548](#)  
[start\\_directory\(\) \(robot.parsing.suitestructure.SuiteStructureVisitor method\), 357](#)  
[start\\_directory\(\) \(robot.running.builder.builders.SuiteStructureParser method\), 504](#)  
[start\\_errors\(\) \(robot.output.xmllogger.XmlLogger method\), 281](#)  
[start\\_errors\(\) \(robot.reporting.outputwriter.OutputWriter method\), 361](#)  
[start\\_errors\(\) \(robot.reporting.xunitwriter.XUnitFileWriter method\), 368](#)  
[start\\_errors\(\) \(robot.result.visitor.ResultVisitor method\), 478](#)  
[start\\_for\(\) \(robot.conf.gatherfailed.GatherFailedSuites method\), 29](#)  
[start\\_for\(\) \(robot.conf.gatherfailed.GatherFailedTests method\), 24](#)  
[start\\_for\(\) \(robot.model.configurer.SuiteConfigurer method\), 201](#)  
[start\\_for\(\) \(robot.model.filter.EmptySuiteRemover method\), 217](#)  
[start\\_for\(\) \(robot.model.filter.Filter method\), 221](#)  
[start\\_for\(\) \(robot.model.modifier.ModelModifier method\), 232](#)  
[start\\_for\(\) \(robot.model.statistics.StatisticsBuilder method\), 237](#)  
[start\\_for\(\) \(robot.model.tagsetter.TagSetter method\), 244](#)  
[start\\_for\(\) \(robot.model.totalstatistics.TotalStatisticsBuilder method\), 255](#)  
[start\\_for\(\) \(robot.model.visitor.SuiteVisitor method\), 260](#)  
[start\\_for\(\) \(robot.output.console.dotted.StatusReporter method\), 265](#)  
[start\\_for\(\) \(robot.output.xmllogger.XmlLogger method\), 279](#)  
[start\\_for\(\) \(robot.reporting.outputwriter.OutputWriter method\), 361](#)  
[start\\_for\(\) \(robot.reporting.xunitwriter.XUnitFileWriter method\), 368](#)  
[start\\_for\(\) \(robot.result.configurer.SuiteConfigurer method\), 374](#)  
[start\\_for\(\) \(robot.result.keywordremover.AllKeywordsRemover method\), 381](#)  
[start\\_for\(\) \(robot.result.keywordremover.ByKeywordNameKeywordRemover method\), 388](#)  
[start\\_for\(\) \(robot.result.keywordremover.ByTagKeywordRemover method\), 393](#)  
[start\\_for\(\) \(robot.result.keywordremover.ForLoopItemsRemover method\), 396](#)  
[start\\_for\(\) \(robot.result.keywordremover.PassedKeywordRemover method\), 385](#)  
[start\\_for\(\) \(robot.result.keywordremover.WaitUntilKeywordSucceeds method\), 406](#)  
[start\\_for\(\) \(robot.result.keywordremover.WarningAndErrorFinder method\), 410](#)  
[start\\_for\(\) \(robot.result.keywordremover.WhileLoopItemsRemover method\), 402](#)  
[start\\_for\(\) \(robot.result.merger.Merger method\), 415](#)  
[start\\_for\(\) \(robot.result.messagefilter.MessageFilter method\), 419](#)  
[start\\_for\(\) \(robot.result.resultbuilder.RemoveKeywords method\), 466](#)  
[start\\_for\(\) \(robot.result.suitetardownfailed.SuiteTeardownFailed method\), 475](#)  
[start\\_for\(\) \(robot.result.suitetardownfailed.SuiteTeardownFailureHandler method\), 471](#)  
[start\\_for\(\) \(robot.result.visitor.ResultVisitor method\), 480](#)  
[start\\_for\(\) \(robot.running.randomizer.Randomizer method\), 542](#)  
[start\\_for\(\) \(robot.running.suiterunner.SuiteRunner method\), 548](#)  
[start\\_for\\_iteration\(\) \(robot.conf.gatherfailed.GatherFailedSuites method\), 29](#)  
[start\\_for\\_iteration\(\) \(robot.conf.gatherfailed.GatherFailedTests method\), 25](#)  
[start\\_for\\_iteration\(\) \(robot.model.configurer.SuiteConfigurer method\), 201](#)  
[start\\_for\\_iteration\(\) \(robot.model.filter.EmptySuiteRemover method\), 217](#)  
[start\\_for\\_iteration\(\) \(robot.model.filter.Filter method\), 221](#)  
[start\\_for\\_iteration\(\) \(robot.model.filter.Filter method\), 221](#)  
[start\\_for\\_iteration\(\) \(robot.model.filter.Filter method\), 221](#)

[\(robot.model.modifier.ModelModifier method\), 232](#)  
[start\\_for\\_iteration\(\) \(robot.model.statistics.StatisticsBuilder method\), 237](#)  
[start\\_for\\_iteration\(\) \(robot.model.tagsetter.TagSetter method\), 244](#)  
[start\\_for\\_iteration\(\) \(robot.model.totalstatistics.TotalStatisticsBuilder method\), 255](#)  
[start\\_for\\_iteration\(\) \(robot.model.visitor.SuiteVisitor method\), 260](#)  
[start\\_for\\_iteration\(\) \(robot.output.console.dotted.StatusReporter method\), 265](#)  
[start\\_for\\_iteration\(\) \(robot.output.xmllogger.XmlLogger method\), 279](#)  
[start\\_for\\_iteration\(\) \(robot.reporting.outputwriter.OutputWriter method\), 361](#)  
[start\\_for\\_iteration\(\) \(robot.reporting.xunitwriter.XUnitFileWriter method\), 368](#)  
[start\\_for\\_iteration\(\) \(robot.result.configurer.SuiteConfigurer method\), 374](#)  
[start\\_for\\_iteration\(\) \(robot.result.keywordremover.AllKeywordsRemover method\), 381](#)  
[start\\_for\\_iteration\(\) \(robot.result.keywordremover.ByNameKeywordRemover method\), 389](#)  
[start\\_for\\_iteration\(\) \(robot.result.keywordremover.ByTagKeywordRemover method\), 394](#)  
[start\\_for\\_iteration\(\) \(robot.result.keywordremover.ForLoopItemsRemover method\), 398](#)  
[start\\_for\\_iteration\(\) \(robot.result.keywordremover.PassedKeywordRemover method\), 385](#)  
[start\\_for\\_iteration\(\) \(robot.result.keywordremover.WaitUntilKeywordSucceedsAndFindsRationale method\), 406](#)  
[start\\_for\\_iteration\(\) \(robot.result.keywordremover.WarningAndErrorFinder method\), 411](#)  
[start\\_for\\_iteration\(\) \(robot.result.keywordremover.WhileLoopItemsRemover method\), 402](#)  
[start\\_for\\_iteration\(\) \(robot.result.merger.Merger method\), 415](#)  
[start\\_for\\_iteration\(\) \(robot.result.messagefilter.MessageFilter method\), 419](#)  
[start\\_for\\_iteration\(\) \(robot.result.resultbuilder.RemoveKeywords method\), 466](#)  
[start\\_for\\_iteration\(\) \(robot.result.suiteteardownfailed.SuiteTeardownFailed method\), 475](#)  
[start\\_for\\_iteration\(\) \(robot.result.suiteteardownfailed.SuiteTeardownFailureHandler method\), 471](#)  
[start\\_for\\_iteration\(\) \(robot.result.visitor.ResultVisitor method\), 480](#)  
[start\\_for\\_iteration\(\) \(robot.running.randomizer.Randomizer method\), 542](#)  
[start\\_for\\_iteration\(\) \(robot.running.suiterunner.SuiteRunner method\), 549](#)  
[start\\_if\(\) \(robot.conf.gatherfailed.GatherFailedSuites method\), 29](#)  
[start\\_if\(\) \(robot.conf.gatherfailed.GatherFailedTests method\), 25](#)  
[start\\_if\(\) \(robot.model.configurer.SuiteConfigurer method\), 201](#)  
[start\\_if\(\) \(robot.model.filter.EmptySuiteRemover method\), 217](#)  
[start\\_if\(\) \(robot.model.filter.Filter method\), 221](#)  
[start\\_if\(\) \(robot.model.modifier.ModelModifier method\), 232](#)  
[start\\_if\(\) \(robot.model.statistics.StatisticsBuilder method\), 237](#)  
[start\\_if\(\) \(robot.model.tagsetter.TagSetter method\), 244](#)  
[start\\_if\(\) \(robot.model.totalstatistics.TotalStatisticsBuilder method\), 255](#)  
[start\\_if\(\) \(robot.model.visitor.SuiteVisitor method\), 260](#)  
[start\\_if\(\) \(robot.output.console.dotted.StatusReporter method\), 266](#)  
[start\\_if\(\) \(robot.output.xmllogger.XmlLogger method\), 278](#)  
[start\\_if\(\) \(robot.reporting.outputwriter.OutputWriter method\), 362](#)  
[start\\_if\(\) \(robot.reporting.xunitwriter.XUnitFileWriter method\), 368](#)  
[start\\_if\(\) \(robot.result.configurer.SuiteConfigurer method\), 374](#)  
[start\\_if\(\) \(robot.result.keywordremover.AllKeywordsRemover method\), 381](#)  
[start\\_if\(\) \(robot.result.keywordremover.ByNameKeywordRemover](#)

<b>Index</b>	<b>707</b>
--------------	------------

(robot.result.suiteteardownfailed.SuiteTeardownFailureHandler.method), 382  
 method), 471  
 start\_if\_branch() (robot.result.visitor.ResultVisitor method), 480  
 start\_if\_branch() (robot.running.randomizer.Randomizer method), 543  
 start\_if\_branch() (robot.running.suiterunner.SuiteRunner method), 549  
 start\_keyword() (robot.conf.gatherfailed.GatherFailedSuites method), 29  
 start\_keyword() (robot.conf.gatherfailed.GatherFailedTests method), 25  
 start\_keyword() (robot.model.configurer.SuiteConfigurer method), 201  
 start\_keyword() (robot.model.filter.EmptySuiteRemover method), 217  
 start\_keyword() (robot.model.filter.Filter method), 221  
 start\_keyword() (robot.model.modifier.ModelModifier method), 232  
 start\_keyword() (robot.model.statistics.StatisticsBuilder method), 237  
 start\_keyword() (robot.model.tagsetter.TagSetter method), 245  
 start\_keyword() (robot.model.totalstatistics.TotalStatisticsBuilder method), 256  
 start\_keyword() (robot.model.visitor.SuiteVisitor method), 259  
 start\_keyword() (robot.output.console.dotted.StatusReporter method), 266  
 start\_keyword() (robot.output.console.verbose.VerboseReporter method), 269  
 start\_keyword() (robot.output.filelogger.FileLogger method), 270  
 start\_keyword() (robot.output.listeners.Listeners method), 272  
 start\_keyword() (robot.output.logger.Logger method), 273  
 start\_keyword() (robot.output.logger.LoggerProxy method), 274  
 start\_keyword() (robot.output.output.Output method), 276  
 start\_keyword() (robot.output.xmllogger.XmlLogger method), 278  
 start\_keyword() (robot.reporting.outputwriter.OutputWriter method), 362  
 start\_keyword() (robot.reporting.xunitwriter.XUnitFileWriter method), 369  
 start\_keyword() (robot.result.configurer.SuiteConfigurer method), 374  
 start\_keyword() (robot.result.keywordremover.AllKeywordsRemover method), 382  
 start\_keyword() (robot.result.keywordremover.ByTagKeywordRemover method), 388  
 start\_keyword() (robot.result.keywordremover.ByTagKeywordRemover method), 392  
 start\_keyword() (robot.result.keywordremover.ForLoopItemsRemover method), 398  
 start\_keyword() (robot.result.keywordremover.PassedKeywordRemover method), 386  
 start\_keyword() (robot.result.keywordremover.WaitUntilKeywordSuccess method), 404  
 start\_keyword() (robot.result.keywordremover.WarningAndErrorFinder method), 409  
 start\_keyword() (robot.result.keywordremover.WhileLoopItemsRemover method), 402  
 start\_keyword() (robot.result.merger.Merger method), 415  
 start\_keyword() (robot.result.messagefilter.MessageFilter method), 417  
 start\_keyword() (robot.result.resultbuilder.RemoveKeywords method), 467  
 start\_keyword() (robot.result.suiteteardownfailed.SuiteTeardownFailureHandler method), 475  
 start\_keyword() (robot.result.suiteteardownfailed.SuiteTeardownFailureHandler method), 471  
 start\_keyword() (robot.result.visitor.ResultVisitor method), 480  
 start\_keyword() (robot.running.randomizer.Randomizer method), 543  
 start\_keyword() (robot.running.suiterunner.SuiteRunner method), 549  
 start\_keyword() (robot.variables.scopes.SetVariables method), 571  
 start\_keyword() (robot.variables.scopes.VariableScopes method), 571  
 start\_loggers (robot.output.logger.Logger attribute), 273  
 start\_message() (robot.conf.gatherfailed.GatherFailedSuites method), 29  
 start\_message() (robot.conf.gatherfailed.GatherFailedTests method), 25  
 start\_message() (robot.model.configurer.SuiteConfigurer method), 201  
 start\_message() (robot.model.filter.EmptySuiteRemover method), 217  
 start\_message() (robot.model.filter.Filter method), 221  
 start\_message() (robot.model.modifier.ModelModifier method), 232  
 start\_message() (robot.model.statistics.StatisticsBuilder method), 237  
 start\_message() (robot.model.tagsetter.TagSetter method), 245  
 start\_message() (robot.model.totalstatistics.TotalStatisticsBuilder method), 256



709

`method`), 475  
`start_return()` (`robot.result.suitetardownfailed.SuiteTardownFailed` `method`), 471  
`start_return()` (`robot.result.visitor.ResultVisitor` `method`), 480  
`start_return()` (`robot.running.randomizer.Randomizer` `method`), 543  
`start_return()` (`robot.running.suiterunner.SuiteRunner` `method`), 549  
`start_splitting_if_needed()`  
   (`robot.reporting.jsbuildingcontext.JsBuildingContext` `method`), 358  
`start_stat()` (`robot.output.xmllogger.XmlLogger` `method`), 281  
`start_stat()` (`robot.reporting.outputwriter.OutputWriter` `method`), 362  
`start_stat()` (`robot.reporting.xunitwriter.XUnitFileWriter` `method`), 369  
`start_stat()` (`robot.result.visitor.ResultVisitor` `method`), 478  
`start_statistics()`  
   (`robot.output.xmllogger.XmlLogger` `method`), 281  
`start_statistics()`  
   (`robot.reporting.outputwriter.OutputWriter` `method`), 362  
`start_statistics()`  
   (`robot.reporting.xunitwriter.XUnitFileWriter` `method`), 369  
`start_statistics()`  
   (`robot.result.visitor.ResultVisitor` `method`), 478  
`start_suite()` (`robot.conf.gatherfailed.GatherFailedSuites` `method`), 27  
`start_suite()` (`robot.conf.gatherfailed.GatherFailedTests` `method`), 25  
`start_suite()` (`robot.model.configurer.SuiteConfigurer` `method`), 201  
`start_suite()` (`robot.model.filter.EmptySuiteRemover` `method`), 217  
`start_suite()` (`robot.model.filter.Filter` `method`), 219  
`start_suite()` (`robot.model.modifier.ModelModifier` `method`), 233  
`start_suite()` (`robot.model.statistics.StatisticsBuilder` `method`), 235  
`start_suite()` (`robot.model.sitestatistics.SuiteStatisticsBuilder` `method`), 241  
`start_suite()` (`robot.model.tagsetter.TagSetter` `method`), 242  
`start_suite()` (`robot.model.totalstatistics.TotalStatisticsBuilder` `method`), 256  
`start_suite()` (`robot.model.visitor.SuiteVisitor` `method`), 259  
`start_suite()` (`robot.output.console.dotted.DottedOutput` `method`), 266  
`start_suite()` (`robot.output.console.dotted.StatusReporter` `method`), 266  
`start_suite()` (`robot.output.console.verbose.VerboseOutput` `method`), 269  
`start_suite()` (`robot.output.filelogger.FileLogger` `method`), 270  
`start_suite()` (`robot.output.logger.Logger` `method`), 273  
`start_suite()` (`robot.output.output.Output` `method`), 276  
`start_suite()` (`robot.output.xmllogger.XmlLogger` `method`), 280  
`start_suite()` (`robot.reporting.outputwriter.OutputWriter` `method`), 362  
`start_suite()` (`robot.reporting.xunitwriter.XUnitFileWriter` `method`), 366  
`start_suite()` (`robot.result.configurer.SuiteConfigurer` `method`), 375  
`start_suite()` (`robot.result.keywordremover.AllKeywordsRemover` `method`), 382  
`start_suite()` (`robot.result.keywordremover.ByNameKeywordRemover` `method`), 390  
`start_suite()` (`robot.result.keywordremover.ByTagKeywordRemover` `method`), 394  
`start_suite()` (`robot.result.keywordremover.ForLoopItemsRemover` `method`), 398  
`start_suite()` (`robot.result.keywordremover.PassedKeywordRemover` `method`), 383  
`start_suite()` (`robot.result.keywordremover.WaitUntilKeywordSucceeds` `method`), 407  
`start_suite()` (`robot.result.keywordremover.WarningAndErrorFinder` `method`), 409  
`start_suite()` (`robot.result.keywordremover.WhileLoopItemsRemover` `method`), 403  
`start_suite()` (`robot.result.merger.Merger` `method`), 413  
`start_suite()` (`robot.result.messagefilter.MessageFilter` `method`), 417  
`start_suite()` (`robot.result.resultbuilder.RemoveKeywords` `method`), 465  
`start_suite()` (`robot.result.suitetardownfailed.SuiteTardownFailed` `method`), 476  
`start_suite()` (`robot.result.suitetardownfailed.SuiteTardownFailure` `method`), 471  
`start_suite()` (`robot.result.visitor.ResultVisitor` `method`), 481  
`start_suite()` (`robot.running.context.ExecutionContexts` `method`), 513  
`start_suite()` (`robot.running.libraryscopes.GlobalScope` `method`), 515  
`start_suite()` (`robot.running.libraryscopes.TestCaseScope` `method`), 516

`start_suite()` (*robot.running.libraryscopes.TestSuiteScope* method), 259  
`start_suite()` (*robot.running.namespaces.Namespace* method), 540  
`start_suite()` (*robot.running.randomizer.Randomizer* method), 541  
`start_suite()` (*robot.running.suiterunner.SuiteRunner* method), 547  
`start_suite()` (*robot.variables.scopes.SetVariables* method), 571  
`start_suite()` (*robot.variables.scopes.VariableScopes* method), 571  
`start_suite_statistics()` (*robot.output.xmllogger.XmlLogger* method), 281  
`start_suite_statistics()` (*robot.reporting.outputwriter.OutputWriter* method), 362  
`start_suite_statistics()` (*robot.reporting.xunitwriter.XUnitFileWriter* method), 369  
`start_suite_statistics()` (*robot.result.visitor.ResultVisitor* method), 478  
`start_tag_statistics()` (*robot.output.xmllogger.XmlLogger* method), 281  
`start_tag_statistics()` (*robot.reporting.outputwriter.OutputWriter* method), 362  
`start_tag_statistics()` (*robot.reporting.xunitwriter.XUnitFileWriter* method), 369  
`start_tag_statistics()` (*robot.result.visitor.ResultVisitor* method), 478  
`start_test()` (*robot.conf.gatherfailed.GatherFailedSuites* method), 29  
`start_test()` (*robot.conf.gatherfailed.GatherFailedTests* method), 25  
`start_test()` (*robot.model.configurer.SuiteConfigurer* method), 202  
`start_test()` (*robot.model.filter.EmptySuiteRemover* method), 218  
`start_test()` (*robot.model.filter.Filter* method), 222  
`start_test()` (*robot.model.modifier.ModelModifier* method), 233  
`start_test()` (*robot.model.statistics.StatisticsBuilder* method), 238  
`start_test()` (*robot.model.tagsetter.TagSetter* method), 245  
`start_test()` (*robot.model.totalstatistics.TotalStatisticsBuilder* method), 256  
`start_test()` (*robot.model.visitor.SuiteVisitor* method), 259  
`start_test()` (*robot.output.console.dotted.StatusReporter* method), 266  
`start_test()` (*robot.output.console.verbose.VerboseOutput* method), 269  
`start_test()` (*robot.output.filelogger.FileLogger* method), 270  
`start_test()` (*robot.output.logger.Logger* method), 273  
`start_test()` (*robot.output.output.Output* method), 276  
`start_test()` (*robot.output.xmllogger.XmlLogger* method), 280  
`start_test()` (*robot.reporting.outputwriter.OutputWriter* method), 362  
`start_test()` (*robot.reporting.xunitwriter.XUnitFileWriter* method), 369  
`start_test()` (*robot.result.configurer.SuiteConfigurer* method), 375  
`start_test()` (*robot.result.keywordremover.AllKeywordsRemover* method), 382  
`start_test()` (*robot.result.keywordremover.ByNameKeywordRemover* method), 390  
`start_test()` (*robot.result.keywordremover.ByTagKeywordRemover* method), 394  
`start_test()` (*robot.result.keywordremover.ForLoopItemsRemover* method), 398  
`start_test()` (*robot.result.keywordremover.PassedKeywordRemover* method), 386  
`start_test()` (*robot.result.keywordremover.WaitUntilKeywordSucceeds* method), 407  
`start_test()` (*robot.result.keywordremover.WarningAndErrorFinder* method), 409  
`start_test()` (*robot.result.keywordremover.WhileLoopItemsRemover* method), 403  
`start_test()` (*robot.result.merger.Merger* method), 415  
`start_test()` (*robot.result.messagefilter.MessageFilter* method), 419  
`start_test()` (*robot.result.resultbuilder.RemoveKeywords* method), 467  
`start_test()` (*robot.result.suitetardownfailed.SuiteTeardownFailed* method), 476  
`start_test()` (*robot.result.suitetardownfailed.SuiteTeardownFailureH* method), 471  
`start_test()` (*robot.result.visitor.ResultVisitor* method), 481  
`start_test()` (*robot.running.libraryscopes.GlobalScope* method), 515  
`start_test()` (*robot.running.libraryscopes.TestCaseScope* method), 515  
`start_test()` (*robot.running.libraryscopes.TestSuiteScope* method), 515  
`start_test()` (*robot.running.namespaces.Namespace*

*method*), 540

`start_test()` (*robot.running.randomizer.Randomizer* *method*), 543

`start_test()` (*robot.running.suiterunner.SuiteRunner* *method*), 549

`start_test()` (*robot.variables.scopes.SetVariables* *method*), 571

`start_test()` (*robot.variables.scopes.VariableScopes* *method*), 571

`start_total_statistics()`  
(*robot.output.xmllogger.XmlLogger* *method*), 281

`start_total_statistics()`  
(*robot.reporting.outputwriter.OutputWriter* *method*), 362

`start_total_statistics()`  
(*robot.reporting.xunitwriter.XUnitFileWriter* *method*), 369

`start_total_statistics()`  
(*robot.result.visitor.ResultVisitor* *method*), 478

`start_try()` (*robot.conf.gatherfailed.GatherFailedSuites* *method*), 29

`start_try()` (*robot.conf.gatherfailed.GatherFailedTests* *method*), 25

`start_try()` (*robot.model.configurer.SuiteConfigurer* *method*), 202

`start_try()` (*robot.model.filter.EmptySuiteRemover* *method*), 218

`start_try()` (*robot.model.filter.Filter* *method*), 222

`start_try()` (*robot.model.modifier.ModelModifier* *method*), 233

`start_try()` (*robot.model.statistics.StatisticsBuilder* *method*), 238

`start_try()` (*robot.model.tagsetter.TagSetter* *method*), 245

`start_try()` (*robot.model.totalstatistics.TotalStatisticsBuilder* *method*), 256

`start_try()` (*robot.model.visitor.SuiteVisitor* *method*), 261

`start_try()` (*robot.output.console.dotted.StatusReporter* *method*), 266

`start_try()` (*robot.output.xmllogger.XmlLogger* *method*), 279

`start_try()` (*robot.reporting.outputwriter.OutputWriter* *method*), 362

`start_try()` (*robot.reporting.xunitwriter.XUnitFileWriter* *method*), 369

`start_try()` (*robot.result.configurer.SuiteConfigurer* *method*), 375

`start_try()` (*robot.result.keywordremover.AllKeywordsRemover* *method*), 382

`start_try()` (*robot.result.keywordremover.ByNameKeywordRemover* *method*), 390

`start_try()` (*robot.result.keywordremover.ByTagKeywordRemover* *method*), 394

`start_try()` (*robot.result.keywordremover.ForLoopItemsRemover* *method*), 398

`start_try()` (*robot.result.keywordremover.PassedKeywordRemover* *method*), 386

`start_try()` (*robot.result.keywordremover.WaitUntilKeywordSucceeds* *method*), 407

`start_try()` (*robot.result.keywordremover.WarningAndErrorFinder* *method*), 411

`start_try()` (*robot.result.keywordremover.WhileLoopItemsRemover* *method*), 403

`start_try()` (*robot.result.merger.Merger* *method*), 415

`start_try()` (*robot.result.messagefilter.MessageFilter* *method*), 420

`start_try()` (*robot.result.resultbuilder.RemoveKeywords* *method*), 467

`start_try()` (*robot.result.suitetearndownfailed.SuiteTearndownFailed* *method*), 476

`start_try()` (*robot.result.suitetearndownfailed.SuiteTearndownFailureHa* *method*), 472

`start_try()` (*robot.result.visitor.ResultVisitor* *method*), 481

`start_try()` (*robot.running.randomizer.Randomizer* *method*), 543

`start_try()` (*robot.running.suiterunner.SuiteRunner* *method*), 549

`start_try_branch()`  
(*robot.conf.gatherfailed.GatherFailedSuites* *method*), 30

`start_try_branch()`  
(*robot.conf.gatherfailed.GatherFailedTests* *method*), 25

`start_try_branch()`  
(*robot.model.configurer.SuiteConfigurer* *method*), 202

`start_try_branch()`  
(*robot.model.filter.EmptySuiteRemover* *method*), 218

`start_try_branch()` (*robot.model.filter.Filter* *method*), 222

`start_try_branch()` (*robot.model.modifier.ModelModifier* *method*), 233

`start_try_branch()` (*robot.model.statistics.StatisticsBuilder* *method*), 238

`start_try_branch()` (*robot.model.tagsetter.TagSetter* *method*), 245

`start_try_branch()` (*robot.model.totalstatistics.TotalStatisticsBuilder* *method*), 256



`start_try_branch()` (*robot.model.visitor.SuiteVisitor* method), 261  
`start_try_branch()` (*robot.output.console.dotted.StatusReporter* method), 266  
`start_try_branch()` (*robot.output.xmllogger.XmlLogger* method), 279  
`start_try_branch()` (*robot.reporting.outputwriter.OutputWriter* method), 362  
`start_try_branch()` (*robot.reporting.xunitwriter.XUnitFileWriter* method), 369  
`start_try_branch()` (*robot.result.configurer.SuiteConfigurer* method), 375  
`start_try_branch()` (*robot.result.keywordremover.AllKeywordsRemover* method), 382  
`start_try_branch()` (*robot.result.keywordremover.ByNameKeywordRemover* method), 390  
`start_try_branch()` (*robot.result.keywordremover.ByTagKeywordRemover* method), 394  
`start_try_branch()` (*robot.result.keywordremover.ForLoopItemsRemover* method), 398  
`start_try_branch()` (*robot.result.keywordremover.PassedKeywordRemover* method), 386  
`start_try_branch()` (*robot.result.keywordremover.WaitUntilKeywordSucceedsRemover* method), 407  
`start_try_branch()` (*robot.result.keywordremover.WarningAndErrorFinder* method), 411  
`start_try_branch()` (*robot.result.keywordremover.WhileLoopItemsRemover* method), 403  
`start_try_branch()` (*robot.result.merger.Merger* method), 416  
`start_try_branch()` (*robot.result.messagefilter.MessageFilter* method), 420  
`start_try_branch()` (*robot.result.resultbuilder.RemoveKeywords* method), 467  
`start_try_branch()` (*robot.result.suiteardownfailed.SuiteTeardownFailed* method), 476  
`start_try_branch()` (*robot.result.suiteardownfailed.SuiteTeardownFailureHandler* method), 472  
`start_try_branch()` (*robot.result.visitor.ResultVisitor* method), 481  
`start_try_branch()` (*robot.running.randomizer.Randomizer* method), 543  
`start_try_branch()` (*robot.running.suiterunner.SuiteRunner* method), 549  
`start_user_keyword()` (*robot.running.namespace.Namespace* method), 540  
`start_while()` (*robot.conf.gatherfailed.GatherFailedSuites* method), 30  
`start_while()` (*robot.conf.gatherfailed.GatherFailedTests* method), 25  
`start_while()` (*robot.model.configurer.SuiteConfigurer* method), 202  
`start_while()` (*robot.model.filter.EmptySuiteRemover* method), 218  
`start_while()` (*robot.model.filter.Filter* method), 222  
`start_while()` (*robot.model.modifier.ModelModifier* method), 233  
`start_while()` (*robot.model.statistics.StatisticsBuilder* method), 238  
`start_while()` (*robot.model.tagsetter.TagSetter* method), 245  
`start_while()` (*robot.model.totalstatistics.TotalStatisticsBuilder* method), 256  
`start_while()` (*robot.model.visitor.SuiteVisitor* method), 261  
`start_while()` (*robot.output.console.dotted.StatusReporter* method), 266  
`start_while()` (*robot.output.xmllogger.XmlLogger* method), 279  
`start_while()` (*robot.reporting.outputwriter.OutputWriter* method), 362  
`start_while()` (*robot.reporting.xunitwriter.XUnitFileWriter* method), 369  
`start_while()` (*robot.result.configurer.SuiteConfigurer* method), 375  
`start_while()` (*robot.result.keywordremover.AllKeywordsRemover* method), 382  
`start_while()` (*robot.result.keywordremover.ByNameKeywordRemover* method), 390  
`start_while()` (*robot.result.keywordremover.ByTagKeywordRemover* method), 394  
`start_while()` (*robot.result.keywordremover.ForLoopItemsRemover* method), 399  
`start_while()` (*robot.result.keywordremover.PassedKeywordRemover* method), 386

`start_while()` (`robot.result.keywordremover.WaitUntilKeywordSucceedsRemover` method), 407

`start_while()` (`robot.result.keywordremover.WarningAndErrorFinder` method), 411

`start_while()` (`robot.result.keywordremover.WhileLoopItemsRemover` method), 400

`start_while()` (`robot.result.merger.Merger` method), 416

`start_while()` (`robot.result.messagefilter.MessageFilter` method), 420

`start_while()` (`robot.result.resultbuilder.RemoveKeywords` method), 467

`start_while()` (`robot.result.suiteteardownfailed.SuiteTeardownFailed` method), 476

`start_while()` (`robot.result.suiteteardownfailed.SuiteTeardownFailureHandler` method), 472

`start_while()` (`robot.result.visitor.ResultVisitor` method), 481

`start_while()` (`robot.running.randomizer.Randomizer` method), 543

`start_while()` (`robot.running.suiterunner.SuiteRunner` method), 549

`start_while_iteration()` (`robot.conf.gatherfailed.GatherFailedSuites` method), 30

`start_while_iteration()` (`robot.conf.gatherfailed.GatherFailedTests` method), 26

`start_while_iteration()` (`robot.model.configurer.SuiteConfigurer` method), 202

`start_while_iteration()` (`robot.model.filter.EmptySuiteRemover` method), 218

`start_while_iteration()` (`robot.model.filter.Filter` method), 222

`start_while_iteration()` (`robot.model.modifier.ModelModifier` method), 233

`start_while_iteration()` (`robot.model.statistics.StatisticsBuilder` method), 238

`start_while_iteration()` (`robot.model.tagsetter.TagSetter` method), 245

`start_while_iteration()` (`robot.model.totalstatistics.TotalStatisticsBuilder` method), 256

`start_while_iteration()` (`robot.model.visitor.SuiteVisitor` method), 262

`start_while_iteration()` (`robot.output.console.dotted.StatusReporter` method), 266

`start_while_iteration()` (`robot.output.xmllogger.XmlLogger` method), 280

`start_while_iteration()` (`robot.reporting.outputwriter.OutputWriter` method), 363

`start_while_iteration()` (`robot.reporting.xunitwriter.XUnitFileWriter` method), 369

`start_while_iteration()` (`robot.result.configurer.SuiteConfigurer` method), 375

`start_while_iteration()` (`robot.result.keywordremover.AllKeywordsRemover` method), 382

`start_while_iteration()` (`robot.result.keywordremover.ByTagKeywordRemover` method), 394

`start_while_iteration()` (`robot.result.keywordremover.ForLoopItemsRemover` method), 399

`start_while_iteration()` (`robot.result.keywordremover.PassedKeywordRemover` method), 386

`start_while_iteration()` (`robot.result.keywordremover.WaitUntilKeywordSucceedsRemover` method), 407

`start_while_iteration()` (`robot.result.keywordremover.WarningAndErrorFinder` method), 411

`start_while_iteration()` (`robot.result.keywordremover.WhileLoopItemsRemover` method), 403

`start_while_iteration()` (`robot.result.merger.Merger` method), 416

`start_while_iteration()` (`robot.result.messagefilter.MessageFilter` method), 420

`start_while_iteration()` (`robot.result.resultbuilder.RemoveKeywords` method), 467

`start_while_iteration()` (`robot.result.suiteteardownfailed.SuiteTeardownFailed` method), 476

`start_while_iteration()` (`robot.result.suiteteardownfailed.SuiteTeardownFailureHandler` method), 472

`start_while_iteration()` (`robot.result.visitor.ResultVisitor` method), 481

`start_while_iteration()`

- (*robot.running.randomizer.Randomizer* method), 543
- `start_while_iteration()` (*robot.running.suiterunner.SuiteRunner* method), 550
- `StartKeywordArguments` (class in *robot.output.listenerarguments*), 271
- `StartSuiteArguments` (class in *robot.output.listenerarguments*), 271
- `StartTestArguments` (class in *robot.output.listenerarguments*), 271
- `starttime` (*robot.result.model.Break* attribute), 452
- `starttime` (*robot.result.model.Continue* attribute), 449
- `starttime` (*robot.result.model.For* attribute), 431
- `starttime` (*robot.result.model.ForIteration* attribute), 429
- `starttime` (*robot.result.model.If* attribute), 440
- `starttime` (*robot.result.model.IfBranch* attribute), 438
- `starttime` (*robot.result.model.Keyword* attribute), 454
- `starttime` (*robot.result.model.Return* attribute), 447
- `starttime` (*robot.result.model.TestCase* attribute), 457
- `starttime` (*robot.result.model.TestSuite* attribute), 460
- `starttime` (*robot.result.model.Try* attribute), 445
- `starttime` (*robot.result.model.TryBranch* attribute), 442
- `starttime` (*robot.result.model.While* attribute), 435
- `starttime` (*robot.result.model.WhileIteration* attribute), 433
- `Stat` (class in *robot.model.stats*), 239
- `stat` (*robot.model.suitestatistics.SuiteStatistics* attribute), 241
- `stat_message` (*robot.result.model.TestSuite* attribute), 463
- `state()` (*robot.libraries.dialogs\_py.InputDialog* method), 144
- `state()` (*robot.libraries.dialogs\_py.MessageDialog* method), 130
- `state()` (*robot.libraries.dialogs\_py.MultipleSelectionDialog* method), 172
- `state()` (*robot.libraries.dialogs\_py.PassFailDialog* method), 186
- `state()` (*robot.libraries.dialogs\_py.SelectionDialog* method), 158
- `Statement` (class in *robot.parsing.model.statements*), 311
- `StatementLexer` (class in *robot.parsing.lexer.statementslexers*), 294
- `Statistics` (class in *robot.model.statistics*), 235
- `statistics` (*robot.result.executionresult.CombinedResult* attribute), 378
- `statistics` (*robot.result.executionresult.Result* attribute), 377
- `statistics` (*robot.result.model.TestSuite* attribute), 462
- `statistics_config` (*robot.conf.settings.RebotSettings* attribute), 33
- `statistics_config` (*robot.conf.settings.RobotSettings* attribute), 32
- `StatisticsBuilder` (class in *robot.model.statistics*), 235
- `StatisticsBuilder` (class in *robot.reporting.jsmodelbuilders*), 358
- `StatisticsHandler` (class in *robot.result.xmllelementhandlers*), 490
- `status` (*robot.errors.BreakLoop* attribute), 579
- `status` (*robot.errors.ContinueLoop* attribute), 579
- `status` (*robot.errors.ExecutionFailed* attribute), 576
- `status` (*robot.errors.ExecutionFailures* attribute), 577
- `status` (*robot.errors.ExecutionPassed* attribute), 578
- `status` (*robot.errors.ExecutionStatus* attribute), 576
- `status` (*robot.errors.HandlerExecutionFailed* attribute), 577
- `status` (*robot.errors.PassExecution* attribute), 578
- `status` (*robot.errors.ReturnFromKeyword* attribute), 579
- `status` (*robot.errors.UserKeywordExecutionFailed* attribute), 577
- `status` (*robot.result.model.Break* attribute), 451
- `status` (*robot.result.model.Continue* attribute), 449
- `status` (*robot.result.model.For* attribute), 431
- `status` (*robot.result.model.ForIteration* attribute), 429
- `status` (*robot.result.model.If* attribute), 440
- `status` (*robot.result.model.IfBranch* attribute), 438
- `status` (*robot.result.model.Keyword* attribute), 454
- `status` (*robot.result.model.Return* attribute), 447
- `status` (*robot.result.model.TestCase* attribute), 457
- `status` (*robot.result.model.TestSuite* attribute), 462
- `status` (*robot.result.model.Try* attribute), 445
- `status` (*robot.result.model.TryBranch* attribute), 442
- `status` (*robot.result.model.While* attribute), 435
- `status` (*robot.result.model.WhileIteration* attribute), 433
- `status` (*robot.running.status.SuiteStatus* attribute), 545
- `status` (*robot.running.status.TestStatus* attribute), 546
- `status()` (*robot.output.console.verbose.VerboseWriter* method), 269
- `status_rc` (*robot.conf.settings.RebotSettings* attribute), 33
- `status_rc` (*robot.conf.settings.RobotSettings* attribute), 32
- `StatusHandler` (class in *robot.result.xmllelementhandlers*), 487
- `StatusMixin` (class in *robot.result.model*), 428
- `StatusReporter` (class in *robot.output.console.dotted*), 264
- `StatusReporter` (class in *robot.running.statusreporter*), 547

`stderr` (*robot.libraries.Process.ExecutionResult attribute*), 93  
`stdout` (*robot.libraries.Process.ExecutionResult attribute*), 93  
`StdoutLogSplitter` (class in *robot.output.stdoutlogsplitter*), 278  
`StoredFinder` (class in *robot.variables.finders*), 569  
`String` (class in *robot.libraries.String*), 97  
`string()` (*robot.reporting.jsbuildingcontext.JsBuildingContext method*), 357  
`StringCache` (class in *robot.reporting.stringcache*), 366  
`StringConverter` (class in *robot.running.arguments.typeconverters*), 496  
`StringDumper` (class in *robot.htmldata.jsonwriter*), 34  
`StringIndex` (class in *robot.reporting.stringcache*), 365  
`strings` (*robot.reporting.jsbuildingcontext.JsBuildingContext attribute*), 357  
`strip()` (*robot.libraries.XML.NamespaceStripper method*), 122  
`strip_string()` (*robot.libraries.String.String method*), 101  
`subtract_date_from_date()` (in module *robot.libraries.DateTime*), 75  
`subtract_time_from_date()` (in module *robot.libraries.DateTime*), 76  
`subtract_time_from_time()` (in module *robot.libraries.DateTime*), 76  
`suite` (*robot.model.statistics.Statistics attribute*), 235  
`suite` (*robot.result.executionresult.Result attribute*), 377  
`suite_config` (*robot.conf.settings.RebotSettings attribute*), 32  
`suite_config` (*robot.conf.settings.RobotSettings attribute*), 31  
`suite_names` (*robot.conf.settings.RebotSettings attribute*), 33  
`suite_names` (*robot.conf.settings.RobotSettings attribute*), 31  
`suite_separator()` (*robot.output.console.verbose.VerboseWriter method*), 269  
`SUITE_SETUP` (*robot.parsing.lexer.tokens.END attribute*), 305  
`SUITE_SETUP` (*robot.parsing.lexer.tokens.EOS attribute*), 303  
`SUITE_SETUP` (*robot.parsing.lexer.tokens.Token attribute*), 299  
`SUITE_TEARDOWN` (*robot.parsing.lexer.tokens.END attribute*), 305  
`SUITE_TEARDOWN` (*robot.parsing.lexer.tokens.EOS attribute*), 303  
`SUITE_TEARDOWN` (*robot.parsing.lexer.tokens.Token attribute*), 299  
`suite_teardown_failed()` (*robot.result.model.TestSuite method*), 463  
`suite_teardown_skipped()` (*robot.result.model.TestSuite method*), 463  
`SuiteBuilder` (class in *robot.reporting.jsmodelbuilders*), 358  
`SuiteBuilder` (class in *robot.running.builder.transformers*), 506  
`SuiteConfigurer` (class in *robot.model.configurer*), 199  
`SuiteConfigurer` (class in *robot.result.configurer*), 372  
`SuiteHandler` (class in *robot.result.xmllelementhandlers*), 483  
`SuiteMessage` (class in *robot.running.status*), 546  
`SuiteNamePatterns` (class in *robot.model.namepatterns*), 235  
`SuiteRunner` (class in *robot.running.suiterunner*), 547  
`suites` (*robot.model.suitestatistics.SuiteStatistics attribute*), 241  
`suites` (*robot.model.testsuite.TestSuite attribute*), 250  
`suites` (*robot.result.model.TestSuite attribute*), 462  
`suites` (*robot.running.model.TestSuite attribute*), 538  
`SuiteSetup` (class in *robot.parsing.model.statements*), 322  
`SuiteStat` (class in *robot.model.stats*), 240  
`SuiteStatistics` (class in *robot.model.suitestatistics*), 241  
`SuiteStatisticsBuilder` (class in *robot.model.suitestatistics*), 241  
`SuiteStatus` (class in *robot.running.status*), 545  
`SuiteStructure` (class in *robot.parsing.suitestructure*), 356  
`SuiteStructureBuilder` (class in *robot.parsing.suitestructure*), 356  
`SuiteStructureParser` (class in *robot.running.builder.builders*), 504  
`SuiteStructureVisitor` (class in *robot.parsing.suitestructure*), 357  
`SuiteTeardown` (class in *robot.parsing.model.statements*), 323  
`SuiteTeardownFailed` (class in *robot.result.suiteteardownfailed*), 473  
`SuiteTeardownFailureHandler` (class in *robot.result.suiteteardownfailed*), 469  
`SuiteVisitor` (class in *robot.model.visitor*), 259  
`SuiteWriter` (class in *robot.reporting.jswriter*), 359  
`supports_kwargs` (*robot.running.dynamicmethods.RunKeyword attribute*), 513  
`switch()` (*robot.utils.connectioncache.ConnectionCache method*), 558  
`switch_connection()`



(*robot.libraries.Telnet.Telnet method*), 106  
*switch\_process()* (*robot.libraries.Process.Process method*), 93  
*system\_decode()* (in module *robot.utils.encoding*), 559  
*system\_encode()* (in module *robot.utils.encoding*), 559

## T

*TableFormatter* (class in *robot.utils.htmlformatters*), 562  
*tag* (*robot.result.xmllelementhandlers.ArgumentHandler attribute*), 489  
*tag* (*robot.result.xmllelementhandlers.ArgumentsHandler attribute*), 489  
*tag* (*robot.result.xmllelementhandlers.AssignHandler attribute*), 489  
*tag* (*robot.result.xmllelementhandlers.BranchHandler attribute*), 485  
*tag* (*robot.result.xmllelementhandlers.BreakHandler attribute*), 486  
*tag* (*robot.result.xmllelementhandlers.ContinueHandler attribute*), 486  
*tag* (*robot.result.xmllelementhandlers.DocHandler attribute*), 487  
*tag* (*robot.result.xmllelementhandlers.ElementHandler attribute*), 483  
*tag* (*robot.result.xmllelementhandlers.ErrorMessageHandler attribute*), 490  
*tag* (*robot.result.xmllelementhandlers.ErrorsHandler attribute*), 490  
*tag* (*robot.result.xmllelementhandlers.ForHandler attribute*), 484  
*tag* (*robot.result.xmllelementhandlers.IfHandler attribute*), 485  
*tag* (*robot.result.xmllelementhandlers.IterationHandler attribute*), 485  
*tag* (*robot.result.xmllelementhandlers.KeywordHandler attribute*), 484  
*tag* (*robot.result.xmllelementhandlers.MessageHandler attribute*), 487  
*tag* (*robot.result.xmllelementhandlers.MetadataHandler attribute*), 487  
*tag* (*robot.result.xmllelementhandlers.MetadataItemHandler attribute*), 488  
*tag* (*robot.result.xmllelementhandlers.MetaHandler attribute*), 488  
*tag* (*robot.result.xmllelementhandlers.PatternHandler attribute*), 486  
*tag* (*robot.result.xmllelementhandlers.ReturnHandler attribute*), 486  
*tag* (*robot.result.xmllelementhandlers.RobotHandler attribute*), 483  
*tag* (*robot.result.xmllelementhandlers.RootHandler attribute*), 483  
*tag* (*robot.result.xmllelementhandlers.StatisticsHandler attribute*), 490  
*tag* (*robot.result.xmllelementhandlers.StatusHandler attribute*), 487  
*tag* (*robot.result.xmllelementhandlers.SuiteHandler attribute*), 483  
*tag* (*robot.result.xmllelementhandlers.TagHandler attribute*), 488  
*tag* (*robot.result.xmllelementhandlers.TagsHandler attribute*), 488  
*tag* (*robot.result.xmllelementhandlers.TestHandler attribute*), 484  
*tag* (*robot.result.xmllelementhandlers.TimeoutHandler attribute*), 488  
*tag* (*robot.result.xmllelementhandlers.TryHandler attribute*), 485  
*tag* (*robot.result.xmllelementhandlers.ValueHandler attribute*), 490  
*tag* (*robot.result.xmllelementhandlers.VarHandler attribute*), 489  
*tag* (*robot.result.xmllelementhandlers.WhileHandler attribute*), 484  
*TagHandler* (class in *robot.result.xmllelementhandlers*), 488  
*TagPattern()* (in module *robot.model.tags*), 242  
*TagPatterns* (class in *robot.model.tags*), 242  
*Tags* (class in *robot.model.tags*), 242  
*Tags* (class in *robot.parsing.model.statements*), 331  
*tags* (*robot.model.keyword.Keyword attribute*), 225  
*tags* (*robot.model.statistics.Statistics attribute*), 235  
*tags* (*robot.model.tagstatistics.TagStatistics attribute*), 247  
*tags* (*robot.model.testcase.TestCase attribute*), 248  
*TAGS* (*robot.parsing.lexer.tokens.END attribute*), 305  
*TAGS* (*robot.parsing.lexer.tokens.EOS attribute*), 303  
*TAGS* (*robot.parsing.lexer.tokens.Token attribute*), 300  
*tags* (*robot.result.model.Break attribute*), 454  
*tags* (*robot.result.model.Continue attribute*), 451  
*tags* (*robot.result.model.For attribute*), 433  
*tags* (*robot.result.model.ForIteration attribute*), 431  
*tags* (*robot.result.model.If attribute*), 442  
*tags* (*robot.result.model.IfBranch attribute*), 440  
*tags* (*robot.result.model.Keyword attribute*), 456  
*tags* (*robot.result.model.Return attribute*), 449  
*tags* (*robot.result.model.TestCase attribute*), 459  
*tags* (*robot.result.model.Try attribute*), 447  
*tags* (*robot.result.model.TryBranch attribute*), 444  
*tags* (*robot.result.model.While attribute*), 438  
*tags* (*robot.result.model.WhileIteration attribute*), 435  
*tags* (*robot.result.model.deprecation.DeprecatedAttributesMixin attribute*), 464

- tags (*robot.running.builder.testsettings.TestSettings* attribute), 506
- tags (*robot.running.model.Keyword* attribute), 519
- tags (*robot.running.model.TestCase* attribute), 534
- tags (*robot.running.model.UserKeyword* attribute), 538
- TagSetter (class in *robot.model.tagsetter*), 242
- TagsHandler (class in *robot.result.xmlelementhandlers*), 488
- TagStat (class in *robot.model.stats*), 240
- TagStatDoc (class in *robot.model.tagstatistics*), 247
- TagStatInfo (class in *robot.model.tagstatistics*), 247
- TagStatistics (class in *robot.model.tagstatistics*), 247
- TagStatisticsBuilder (class in *robot.model.tagstatistics*), 247
- TagStatLink (class in *robot.model.tagstatistics*), 247
- take\_screenshot () (*robot.libraries.Screenshot.Screenshot* method), 96
- take\_screenshot\_without\_embedding () (*robot.libraries.Screenshot.Screenshot* method), 97
- tasks (*robot.parsing.model.blocks.TestCaseSection* attribute), 307
- Teardown (class in *robot.parsing.model.statements*), 330
- TEARDOWN (*robot.model.body.BodyItem* attribute), 193
- TEARDOWN (*robot.model.control.Break* attribute), 214
- TEARDOWN (*robot.model.control.Continue* attribute), 213
- TEARDOWN (*robot.model.control.For* attribute), 204
- TEARDOWN (*robot.model.control.If* attribute), 208
- TEARDOWN (*robot.model.control.IfBranch* attribute), 207
- TEARDOWN (*robot.model.control.Return* attribute), 212
- TEARDOWN (*robot.model.control.Try* attribute), 211
- TEARDOWN (*robot.model.control.TryBranch* attribute), 209
- TEARDOWN (*robot.model.control.While* attribute), 205
- TEARDOWN (*robot.model.keyword.Keyword* attribute), 226
- teardown (*robot.model.keyword.Keyword* attribute), 224
- teardown (*robot.model.keyword.Keywords* attribute), 226
- TEARDOWN (*robot.model.message.Message* attribute), 228
- teardown (*robot.model.testcase.TestCase* attribute), 248
- teardown (*robot.model.testsuite.TestSuite* attribute), 251
- TEARDOWN (*robot.output.loggerhelper.Message* attribute), 275
- TEARDOWN (*robot.parsing.lexer.tokens.END* attribute), 305
- TEARDOWN (*robot.parsing.lexer.tokens.EOS* attribute), 303
- TEARDOWN (*robot.parsing.lexer.tokens.Token* attribute), 300
- TEARDOWN (*robot.result.model.Break* attribute), 452
- TEARDOWN (*robot.result.model.Continue* attribute), 450
- TEARDOWN (*robot.result.model.For* attribute), 432
- TEARDOWN (*robot.result.model.ForIteration* attribute), 429
- TEARDOWN (*robot.result.model.If* attribute), 441
- TEARDOWN (*robot.result.model.IfBranch* attribute), 439
- TEARDOWN (*robot.result.model.Keyword* attribute), 455
- teardown (*robot.result.model.Keyword* attribute), 456
- TEARDOWN (*robot.result.model.Message* attribute), 427
- TEARDOWN (*robot.result.model.Return* attribute), 448
- teardown (*robot.result.model.TestCase* attribute), 459
- teardown (*robot.result.model.TestSuite* attribute), 462
- TEARDOWN (*robot.result.model.Try* attribute), 445
- TEARDOWN (*robot.result.model.TryBranch* attribute), 443
- TEARDOWN (*robot.result.model.While* attribute), 436
- TEARDOWN (*robot.result.model.WhileIteration* attribute), 434
- teardown (*robot.running.builder.testsettings.TestDefaults* attribute), 505
- teardown (*robot.running.builder.testsettings.TestSettings* attribute), 506
- TEARDOWN (*robot.running.model.Break* attribute), 531
- TEARDOWN (*robot.running.model.Continue* attribute), 530
- TEARDOWN (*robot.running.model.For* attribute), 520
- TEARDOWN (*robot.running.model.If* attribute), 525
- TEARDOWN (*robot.running.model.IfBranch* attribute), 523
- TEARDOWN (*robot.running.model.Keyword* attribute), 518
- teardown (*robot.running.model.Keyword* attribute), 519
- TEARDOWN (*robot.running.model.Return* attribute), 529
- teardown (*robot.running.model.TestCase* attribute), 534
- teardown (*robot.running.model.TestSuite* attribute), 538
- TEARDOWN (*robot.running.model.Try* attribute), 527
- TEARDOWN (*robot.running.model.TryBranch* attribute), 526
- teardown (*robot.running.model.UserKeyword* attribute), 538
- TEARDOWN (*robot.running.model.While* attribute), 522
- teardown\_allowed (*robot.running.status.Exit* attribute), 545
- teardown\_allowed (*robot.running.status.SuiteStatus* attribute), 545
- teardown\_allowed (*robot.running.status.TestStatus*

*attribute*), 546  
teardown\_executed() (*robot.running.status.SuiteStatus method*), 545  
teardown\_executed() (*robot.running.status.TestStatus method*), 546  
teardown\_message(*robot.running.status.ParentMessage attribute*), 546  
teardown\_message(*robot.running.status.SuiteMessage attribute*), 546  
teardown\_message(*robot.running.status.TestMessage attribute*), 546  
teardown\_skipped\_message (*robot.running.status.ParentMessage attribute*), 546  
teardown\_skipped\_message (*robot.running.status.SuiteMessage attribute*), 546  
teardown\_skipped\_message (*robot.running.status.TestMessage attribute*), 546  
Telnet (*class in robot.libraries.Telnet*), 103  
TelnetConnection (*class in robot.libraries.Telnet*), 106  
Template (*class in robot.parsing.model.statements*), 332  
TEMPLATE (*robot.parsing.lexer.tokens.END attribute*), 305  
TEMPLATE (*robot.parsing.lexer.tokens.EOS attribute*), 303  
TEMPLATE (*robot.parsing.lexer.tokens.Token attribute*), 300  
template (*robot.running.builder.testsettings.TestSettings attribute*), 506  
template (*robot.running.model.TestCase attribute*), 532  
template\_set (*robot.parsing.lexer.context.KeywordContext attribute*), 290  
template\_set (*robot.parsing.lexer.context.TestCaseContext attribute*), 289  
template\_set (*robot.parsing.lexer.settings.TestCaseSettings attribute*), 293  
TemplateArguments (*class in robot.parsing.model.statements*), 336  
TerminalEmulator (*class in robot.libraries.Telnet*), 111  
terminate\_all\_processes() (*robot.libraries.Process.Process method*), 92  
terminate\_process() (*robot.libraries.Process.Process method*), 91  
TERMINATE\_TIMEOUT (*robot.libraries.Process.Process attribute*), 90  
test() (*robot.libraries.Screenshot.ScreenshotTaker method*), 97  
test\_case() (*robot.parsing.lexer.sections.InitFileSections method*), 292  
test\_case() (*robot.parsing.lexer.sections.ResourceFileSections method*), 291  
test\_case() (*robot.parsing.lexer.sections.Sections method*), 291  
test\_case() (*robot.parsing.lexer.sections.TestCaseFileSections method*), 291  
test\_case\_context() (*robot.parsing.lexer.context.TestCaseFileContext method*), 288  
TEST\_CASE\_FILE\_TYPE (*robot.running.handlerstore.HandlerStore attribute*), 514  
TEST\_CASE\_FILE\_TYPE (*robot.running.userkeyword.UserLibrary attribute*), 552  
test\_case\_markers (*robot.parsing.lexer.sections.InitFileSections attribute*), 292  
test\_case\_markers (*robot.parsing.lexer.sections.ResourceFileSections attribute*), 291  
test\_case\_markers (*robot.parsing.lexer.sections.Sections attribute*), 290  
test\_case\_markers (*robot.parsing.lexer.sections.TestCaseFileSections attribute*), 291  
test\_case\_section() (*robot.parsing.lexer.context.FileContext method*), 288  
test\_case\_section() (*robot.parsing.lexer.context.InitFileContext method*), 289  
test\_case\_section() (*robot.parsing.lexer.context.ResourceFileContext method*), 289  
test\_case\_section() (*robot.parsing.lexer.context.TestCaseFileContext method*), 289  
test\_class (*robot.model.testsuite.TestSuite attribute*), 250  
test\_class (*robot.result.model.TestSuite attribute*), 460  
test\_class (*robot.running.model.TestSuite attribute*), 534  
test\_count (*robot.model.testsuite.TestSuite attribute*), 251  
test\_count (*robot.result.model.TestSuite attribute*),

- 462
- `test_count` (*robot.running.model.TestSuite* attribute), 538
- `test_failed()` (*robot.running.status.TestStatus* method), 545
- `test_names` (*robot.conf.settings.RebotSettings* attribute), 33
- `test_names` (*robot.conf.settings.RobotSettings* attribute), 31
- `test_or_task()` (in module *robot.utils.misc*), 566
- `test_separator()` (*robot.output.console.verbose.VerboseWriter* method), 269
- `TEST_SETUP` (*robot.parsing.lexer.tokens.END* attribute), 305
- `TEST_SETUP` (*robot.parsing.lexer.tokens.EOS* attribute), 303
- `TEST_SETUP` (*robot.parsing.lexer.tokens.Token* attribute), 300
- `test_skipped()` (*robot.running.status.TestStatus* method), 545
- `TEST_TEARDOWN` (*robot.parsing.lexer.tokens.END* attribute), 305
- `TEST_TEARDOWN` (*robot.parsing.lexer.tokens.EOS* attribute), 303
- `TEST_TEARDOWN` (*robot.parsing.lexer.tokens.Token* attribute), 300
- `TEST_TEMPLATE` (*robot.parsing.lexer.tokens.END* attribute), 305
- `TEST_TEMPLATE` (*robot.parsing.lexer.tokens.EOS* attribute), 303
- `TEST_TEMPLATE` (*robot.parsing.lexer.tokens.Token* attribute), 300
- `TEST_TIMEOUT` (*robot.parsing.lexer.tokens.END* attribute), 305
- `TEST_TIMEOUT` (*robot.parsing.lexer.tokens.EOS* attribute), 303
- `TEST_TIMEOUT` (*robot.parsing.lexer.tokens.Token* attribute), 300
- `TestBuilder` (class in *robot.reporting.jsmodelbuilders*), 358
- `TestCase` (class in *robot.model.testcase*), 247
- `TestCase` (class in *robot.parsing.model.blocks*), 308
- `TestCase` (class in *robot.result.model*), 457
- `TestCase` (class in *robot.running.model*), 532
- `TESTCASE_HEADER` (*robot.parsing.lexer.tokens.END* attribute), 305
- `TESTCASE_HEADER` (*robot.parsing.lexer.tokens.EOS* attribute), 303
- `TESTCASE_HEADER` (*robot.parsing.lexer.tokens.Token* attribute), 299
- `TESTCASE_NAME` (*robot.parsing.lexer.tokens.END* attribute), 305
- `TESTCASE_NAME` (*robot.parsing.lexer.tokens.EOS* attribute), 303
- `TESTCASE_NAME` (*robot.parsing.lexer.tokens.Token* attribute), 299
- `TestCaseBuilder` (class in *robot.running.builder.transformers*), 507
- `TestCaseContext` (class in *robot.parsing.lexer.context*), 289
- `TestCaseFileContext` (class in *robot.parsing.lexer.context*), 288
- `TestCaseFileSections` (class in *robot.parsing.lexer.sections*), 291
- `TestCaseFileSettings` (class in *robot.parsing.lexer.settings*), 292
- `TestCaseLexer` (class in *robot.parsing.lexer.blocklexers*), 286
- `TestCaseName` (class in *robot.parsing.model.statements*), 328
- `TestCaseParser` (class in *robot.parsing.parser.blockparsers*), 353
- `TestCases` (class in *robot.model.testcase*), 249
- `TestCaseScope` (class in *robot.running.libraryscopes*), 515
- `TestCaseSection` (class in *robot.parsing.model.blocks*), 307
- `TestCaseSectionHeaderLexer` (class in *robot.parsing.lexer.statementlexers*), 295
- `TestCaseSectionLexer` (class in *robot.parsing.lexer.blocklexers*), 284
- `TestCaseSectionParser` (class in *robot.parsing.parser.fileparser*), 355
- `TestCaseSettings` (class in *robot.parsing.lexer.settings*), 293
- `TestDefaults` (class in *robot.running.builder.testsettings*), 505
- `TestDoc` (class in *robot.testdoc*), 585
- `testdoc()` (in module *robot.testdoc*), 586
- `testdoc_cli()` (in module *robot.testdoc*), 586
- `TestdocModelWriter` (class in *robot.testdoc*), 585
- `TestHandler` (class in *robot.result.xmllelementhandlers*), 484
- `TestLibrary()` (in module *robot.running.testlibraries*), 551
- `TestMessage` (class in *robot.running.status*), 546
- `TestNamePatterns` (class in *robot.model.namepatterns*), 235
- `TestOrKeywordLexer` (class in *robot.parsing.lexer.blocklexers*), 286
- `TestOrKeywordSettingLexer` (class in *robot.parsing.lexer.statementlexers*), 296
- `tests` (*robot.model.testsuite.TestSuite* attribute), 250
- `tests` (*robot.result.model.TestSuite* attribute), 462
- `tests` (*robot.running.model.TestSuite* attribute), 538
- `TestSettings` (class in *robot.running.builder.testsettings*), 506
- `TestSetup` (class in *robot.parsing.model.statements*),



- 323
- TestStatus (class in *robot.running.status*), 545
- TestSuite (class in *robot.model.testsuite*), 250
- TestSuite (class in *robot.result.model*), 459
- TestSuite (class in *robot.running.model*), 534
- TestSuiteBuilder (class in *robot.running.builder.builders*), 504
- TestSuiteFactory () (in module *robot.testdoc*), 585
- TestSuites (class in *robot.model.testsuite*), 252
- TestSuiteScope (class in *robot.running.libraryscopes*), 515
- TestTeardown (class in *robot.parsing.model.statements*), 324
- TestTemplate (class in *robot.parsing.model.statements*), 325
- TestTimeout (class in *robot.parsing.model.statements*), 326
- TestTimeout (class in *robot.running.timeouts*), 510
- time\_left () (*robot.running.timeouts.KeywordTimeout* method), 510
- time\_left () (*robot.running.timeouts.TestTimeout* method), 510
- timed\_out () (*robot.running.timeouts.KeywordTimeout* method), 510
- timed\_out () (*robot.running.timeouts.TestTimeout* method), 510
- TimeDeltaConverter (class in *robot.running.arguments.typeconverters*), 499
- Timeout (class in *robot.parsing.model.statements*), 333
- Timeout (class in *robot.running.timeouts.posix*), 511
- Timeout (class in *robot.running.timeouts.windows*), 511
- timeout (*robot.errors.BreakLoop* attribute), 579
- timeout (*robot.errors.ContinueLoop* attribute), 579
- timeout (*robot.errors.ExecutionFailed* attribute), 576
- timeout (*robot.errors.ExecutionFailures* attribute), 577
- timeout (*robot.errors.ExecutionPassed* attribute), 578
- timeout (*robot.errors.ExecutionStatus* attribute), 576
- timeout (*robot.errors.HandlerExecutionFailed* attribute), 577
- timeout (*robot.errors.PassExecution* attribute), 578
- timeout (*robot.errors.ReturnFromKeyword* attribute), 579
- timeout (*robot.errors.UserKeywordExecutionFailed* attribute), 577
- timeout (*robot.model.keyword.Keyword* attribute), 224
- timeout (*robot.model.testcase.TestCase* attribute), 248
- TIMEOUT (*robot.parsing.lexer.tokens.END* attribute), 305
- TIMEOUT (*robot.parsing.lexer.tokens.EOS* attribute), 303
- TIMEOUT (*robot.parsing.lexer.tokens.Token* attribute), 300
- timeout (*robot.result.model.Break* attribute), 454
- timeout (*robot.result.model.Continue* attribute), 451
- timeout (*robot.result.model.For* attribute), 433
- timeout (*robot.result.model.ForIteration* attribute), 431
- timeout (*robot.result.model.If* attribute), 442
- timeout (*robot.result.model.IfBranch* attribute), 440
- timeout (*robot.result.model.Keyword* attribute), 457
- timeout (*robot.result.model.Return* attribute), 449
- timeout (*robot.result.model.TestCase* attribute), 459
- timeout (*robot.result.model.Try* attribute), 447
- timeout (*robot.result.model.TryBranch* attribute), 444
- timeout (*robot.result.model.While* attribute), 438
- timeout (*robot.result.model.WhileIteration* attribute), 435
- timeout (*robot.result.model.deprecation.DeprecatedAttributesMixin* attribute), 464
- timeout (*robot.running.builder.testsettings.TestDefaults* attribute), 506
- timeout (*robot.running.builder.testsettings.TestSettings* attribute), 506
- timeout (*robot.running.model.Keyword* attribute), 520
- timeout (*robot.running.model.TestCase* attribute), 534
- TimeoutError, 575
- TimeoutHandler (class in *robot.result.xmllelementhandlers*), 488
- TimeoutHTTPSTransport (class in *robot.libraries.Remote*), 95
- TimeoutHTTPTransport (class in *robot.libraries.Remote*), 94
- timestamp (*robot.model.message.Message* attribute), 227
- timestamp (*robot.output.loggerhelper.Message* attribute), 276
- timestamp (*robot.result.model.Message* attribute), 428
- timestamp () (*robot.reporting.jsbuildingcontext.JsBuildingContext* method), 357
- title () (*robot.libraries.dialogs\_py.InputDialog* method), 144
- title () (*robot.libraries.dialogs\_py.MessageDialog* method), 131
- title () (*robot.libraries.dialogs\_py.MultipleSelectionDialog* method), 172
- title () (*robot.libraries.dialogs\_py.PassFailDialog* method), 186
- title () (*robot.libraries.dialogs\_py.SelectionDialog* method), 158
- tk\_bisque () (*robot.libraries.dialogs\_py.InputDialog* method), 144
- tk\_bisque () (*robot.libraries.dialogs\_py.MessageDialog* method), 131
- tk\_bisque () (*robot.libraries.dialogs\_py.MultipleSelectionDialog* method), 172
- tk\_bisque () (*robot.libraries.dialogs\_py.PassFailDialog*

*method*), 186

`tk_bisque()` (*robot.libraries.dialogs\_py.SelectionDialog*  
*method*), 158

`tk_focusFollowsMouse()`  
(*robot.libraries.dialogs\_py.InputDialog*  
*method*), 145

`tk_focusFollowsMouse()`  
(*robot.libraries.dialogs\_py.MessageDialog*  
*method*), 131

`tk_focusFollowsMouse()`  
(*robot.libraries.dialogs\_py.MultipleSelectionDialog*  
*method*), 173

`tk_focusFollowsMouse()`  
(*robot.libraries.dialogs\_py.PassFailDialog*  
*method*), 187

`tk_focusFollowsMouse()`  
(*robot.libraries.dialogs\_py.SelectionDialog*  
*method*), 159

`tk_focusNext()` (*robot.libraries.dialogs\_py.InputDialog*  
*method*), 145

`tk_focusNext()` (*robot.libraries.dialogs\_py.MessageDialog*  
*method*), 131

`tk_focusNext()` (*robot.libraries.dialogs\_py.MultipleSelectionDialog*  
*method*), 173

`tk_focusNext()` (*robot.libraries.dialogs\_py.PassFailDialog*  
*method*), 187

`tk_focusNext()` (*robot.libraries.dialogs\_py.SelectionDialog*  
*method*), 159

`tk_focusPrev()` (*robot.libraries.dialogs\_py.InputDialog*  
*method*), 145

`tk_focusPrev()` (*robot.libraries.dialogs\_py.MessageDialog*  
*method*), 131

`tk_focusPrev()` (*robot.libraries.dialogs\_py.MultipleSelectionDialog*  
*method*), 173

`tk_focusPrev()` (*robot.libraries.dialogs\_py.PassFailDialog*  
*method*), 187

`tk_focusPrev()` (*robot.libraries.dialogs\_py.SelectionDialog*  
*method*), 159

`tk_setPalette()` (*robot.libraries.dialogs\_py.InputDialog*  
*method*), 145

`tk_setPalette()` (*robot.libraries.dialogs\_py.MessageDialog*  
*method*), 131

`tk_setPalette()` (*robot.libraries.dialogs\_py.MultipleSelectionDialog*  
*method*), 173

`tk_setPalette()` (*robot.libraries.dialogs\_py.PassFailDialog*  
*method*), 187

`tk_setPalette()` (*robot.libraries.dialogs\_py.SelectionDialog*  
*method*), 159

`tk_strictMotif()` (*robot.libraries.dialogs\_py.InputDialog*  
*method*), 145

`tk_strictMotif()` (*robot.libraries.dialogs\_py.MessageDialog*  
*method*), 131

`tk_strictMotif()` (*robot.libraries.dialogs\_py.MultipleSelectionDialog*  
*method*), 173

`tk_strictMotif()` (*robot.libraries.dialogs\_py.PassFailDialog*  
*method*), 187

`tk_strictMotif()` (*robot.libraries.dialogs\_py.SelectionDialog*  
*method*), 159

`tk_raise()` (*robot.libraries.dialogs\_py.InputDialog*  
*method*), 145

`tk_raise()` (*robot.libraries.dialogs\_py.MessageDialog*  
*method*), 131

`tk_raise()` (*robot.libraries.dialogs\_py.MultipleSelectionDialog*  
*method*), 173

`tk_raise()` (*robot.libraries.dialogs\_py.PassFailDialog*  
*method*), 187

`tk_raise()` (*robot.libraries.dialogs\_py.SelectionDialog*  
*method*), 159

`to_bytes()` (*robot.reporting.stringcache.StringIndex*  
*method*), 365

`to_dictionary()` (*robot.libdocpkg.datatypes.EnumMember*  
*method*), 36

`to_dictionary()` (*robot.libdocpkg.datatypes.TypedDictItem*  
*method*), 36

`to_logdictionary()` (*robot.libdocpkg.datatypes.TypeDoc*  
*method*), 36

`to_logdictionary()` (*robot.libdocpkg.model.KeywordDoc*  
*method*), 38

`to_logdictionary()` (*robot.libdocpkg.model.LibraryDoc*  
*method*), 38

`to_logjson()` (*robot.libdocpkg.model.LibraryDoc*  
*method*), 38

`token` (class in *robot.parsing.lexer.tokens*), 299

`token_type` (*robot.parsing.lexer.statemntlexers.BreakLexer*  
*attribute*), 299

`token_type` (*robot.parsing.lexer.statemntlexers.CommentLexer*  
*attribute*), 295

`token_type` (*robot.parsing.lexer.statemntlexers.CommentSectionHeaderLexer*  
*attribute*), 295

`token_type` (*robot.parsing.lexer.statemntlexers.ContinueLexer*  
*attribute*), 298

`token_type` (*robot.parsing.lexer.statemntlexers.ElseHeaderLexer*  
*attribute*), 297

`token_type` (*robot.parsing.lexer.statemntlexers.ElseIfHeaderLexer*  
*attribute*), 297

`token_type` (*robot.parsing.lexer.statemntlexers.EndLexer*  
*attribute*), 298

`token_type` (*robot.parsing.lexer.statemntlexers.ErrorSectionHeaderLexer*  
*attribute*), 295

`token_type` (*robot.parsing.lexer.statemntlexers.ExceptHeaderLexer*  
*attribute*), 297

`token_type` (*robot.parsing.lexer.statemntlexers.FinallyHeaderLexer*  
*attribute*), 298

`token_type` (*robot.parsing.lexer.statemntlexers.ForHeaderLexer*  
*attribute*), 296

`token_type` (*robot.parsing.lexer.statemntlexers.IfHeaderLexer*  
*attribute*), 297

`token_type` (*robot.parsing.lexer.statemntlexers.InlineIfHeaderLexer*  
*attribute*), 297

<b>Index</b>	<b>723</b>
--------------	------------

- [TRY \(\*robot.result.model.Keyword\* attribute\), 455](#)
- [TRY \(\*robot.result.model.Message\* attribute\), 427](#)
- [TRY \(\*robot.result.model.Return\* attribute\), 448](#)
- [TRY \(\*robot.result.model.Try\* attribute\), 445](#)
- [TRY \(\*robot.result.model.TryBranch\* attribute\), 443](#)
- [TRY \(\*robot.result.model.While\* attribute\), 436](#)
- [TRY \(\*robot.result.model.WhileIteration\* attribute\), 434](#)
- [TRY \(\*robot.running.model.Break\* attribute\), 531](#)
- [TRY \(\*robot.running.model.Continue\* attribute\), 530](#)
- [TRY \(\*robot.running.model.For\* attribute\), 520](#)
- [TRY \(\*robot.running.model.If\* attribute\), 525](#)
- [TRY \(\*robot.running.model.IfBranch\* attribute\), 523](#)
- [TRY \(\*robot.running.model.Keyword\* attribute\), 518](#)
- [TRY \(\*robot.running.model.Return\* attribute\), 529](#)
- [TRY \(\*robot.running.model.Try\* attribute\), 527](#)
- [TRY \(\*robot.running.model.TryBranch\* attribute\), 526](#)
- [TRY \(\*robot.running.model.While\* attribute\), 522](#)
- [try\\_branch \(\*robot.model.control.Try\* attribute\), 210](#)
- [try\\_branch \(\*robot.result.model.Try\* attribute\), 447](#)
- [try\\_branch \(\*robot.running.model.Try\* attribute\), 528](#)
- [try\\_class \(\*robot.model.body.BaseBody\* attribute\), 194](#)
- [try\\_class \(\*robot.model.body.Body\* attribute\), 197](#)
- [try\\_class \(\*robot.model.body.Branches\* attribute\), 199](#)
- [try\\_class \(\*robot.result.model.Body\* attribute\), 423](#)
- [try\\_class \(\*robot.result.model.Branches\* attribute\), 425](#)
- [try\\_class \(\*robot.result.model.Iterations\* attribute\), 426](#)
- [try\\_class \(\*robot.running.model.Body\* attribute\), 517](#)
- [TRY\\_EXCEPT\\_ROOT \(\*robot.model.body.BodyItem\* attribute\), 193](#)
- [TRY\\_EXCEPT\\_ROOT \(\*robot.model.control.Break\* attribute\), 214](#)
- [TRY\\_EXCEPT\\_ROOT \(\*robot.model.control.Continue\* attribute\), 213](#)
- [TRY\\_EXCEPT\\_ROOT \(\*robot.model.control.For\* attribute\), 204](#)
- [TRY\\_EXCEPT\\_ROOT \(\*robot.model.control.If\* attribute\), 208](#)
- [TRY\\_EXCEPT\\_ROOT \(\*robot.model.control.IfBranch\* attribute\), 207](#)
- [TRY\\_EXCEPT\\_ROOT \(\*robot.model.control.Return\* attribute\), 212](#)
- [TRY\\_EXCEPT\\_ROOT \(\*robot.model.control.Try\* attribute\), 211](#)
- [TRY\\_EXCEPT\\_ROOT \(\*robot.model.control.TryBranch\* attribute\), 209](#)
- [TRY\\_EXCEPT\\_ROOT \(\*robot.model.control.While\* attribute\), 205](#)
- [TRY\\_EXCEPT\\_ROOT \(\*robot.model.keyword.Keyword\* attribute\), 226](#)
- [TRY\\_EXCEPT\\_ROOT \(\*robot.model.message.Message\* attribute\), 228](#)
- [TRY\\_EXCEPT\\_ROOT \(\*robot.output.loggerhelper.Message\* attribute\), 275](#)
- [TRY\\_EXCEPT\\_ROOT \(\*robot.result.model.Break\* attribute\), 452](#)
- [TRY\\_EXCEPT\\_ROOT \(\*robot.result.model.Continue\* attribute\), 450](#)
- [TRY\\_EXCEPT\\_ROOT \(\*robot.result.model.For\* attribute\), 432](#)
- [TRY\\_EXCEPT\\_ROOT \(\*robot.result.model.ForIteration\* attribute\), 429](#)
- [TRY\\_EXCEPT\\_ROOT \(\*robot.result.model.If\* attribute\), 441](#)
- [TRY\\_EXCEPT\\_ROOT \(\*robot.result.model.IfBranch\* attribute\), 439](#)
- [TRY\\_EXCEPT\\_ROOT \(\*robot.result.model.Keyword\* attribute\), 455](#)
- [TRY\\_EXCEPT\\_ROOT \(\*robot.result.model.Message\* attribute\), 427](#)
- [TRY\\_EXCEPT\\_ROOT \(\*robot.result.model.Return\* attribute\), 448](#)
- [TRY\\_EXCEPT\\_ROOT \(\*robot.result.model.Try\* attribute\), 445](#)
- [TRY\\_EXCEPT\\_ROOT \(\*robot.result.model.TryBranch\* attribute\), 443](#)
- [TRY\\_EXCEPT\\_ROOT \(\*robot.result.model.While\* attribute\), 436](#)
- [TRY\\_EXCEPT\\_ROOT \(\*robot.result.model.WhileIteration\* attribute\), 434](#)
- [TRY\\_EXCEPT\\_ROOT \(\*robot.running.model.Break\* attribute\), 531](#)
- [TRY\\_EXCEPT\\_ROOT \(\*robot.running.model.Continue\* attribute\), 530](#)
- [TRY\\_EXCEPT\\_ROOT \(\*robot.running.model.For\* attribute\), 520](#)
- [TRY\\_EXCEPT\\_ROOT \(\*robot.running.model.If\* attribute\), 525](#)
- [TRY\\_EXCEPT\\_ROOT \(\*robot.running.model.IfBranch\* attribute\), 523](#)
- [TRY\\_EXCEPT\\_ROOT \(\*robot.running.model.Keyword\* attribute\), 518](#)
- [TRY\\_EXCEPT\\_ROOT \(\*robot.running.model.Return\* attribute\), 529](#)
- [TRY\\_EXCEPT\\_ROOT \(\*robot.running.model.Try\* attribute\), 527](#)
- [TRY\\_EXCEPT\\_ROOT \(\*robot.running.model.TryBranch\* attribute\), 526](#)
- [TRY\\_EXCEPT\\_ROOT \(\*robot.running.model.While\* attribute\), 522](#)
- [TryBranch \(class in \*robot.model.control\*\), 209](#)
- [TryBranch \(class in \*robot.result.model\*\), 442](#)
- [TryBranch \(class in \*robot.running.model\*\), 525](#)
- [TryBuilder \(class in \*robot.running.builder.transformers\*\), 509](#)
- [TryHandler \(class in \*robot.running.builder.transformers\*\), 509](#)



- robot.result.xmlelementhandlers*), 485
- TryHeader (class in *robot.parsing.model.statements*), 343
- TryHeaderLexer (class in *robot.parsing.lexer.statementslexers*), 297
- TryLexer (class in *robot.parsing.lexer.blocklexers*), 288
- TryParser (class in *robot.parsing.parser.blockparsers*), 354
- TryRunner (class in *robot.running.bodyrunner*), 512
- TupleConverter (class in *robot.running.arguments.typeconverters*), 501
- TupleListDumper (class in *robot.htmldata.jsonwriter*), 35
- type (*robot.model.body.BodyItem* attribute), 193
- type (*robot.model.control.Break* attribute), 215
- type (*robot.model.control.Continue* attribute), 213
- type (*robot.model.control.For* attribute), 203
- type (*robot.model.control.If* attribute), 207
- type (*robot.model.control.IfBranch* attribute), 206
- type (*robot.model.control.Return* attribute), 211
- type (*robot.model.control.Try* attribute), 210
- type (*robot.model.control.TryBranch* attribute), 209
- type (*robot.model.control.While* attribute), 205
- type (*robot.model.keyword.Keyword* attribute), 224
- type (*robot.model.message.Message* attribute), 227
- type (*robot.model.stats.CombinedTagStat* attribute), 241
- type (*robot.model.stats.SuiteStat* attribute), 240
- type (*robot.model.stats.TagStat* attribute), 241
- type (*robot.model.stats.TotalStat* attribute), 240
- type (*robot.output.loggerhelper.Message* attribute), 276
- type (*robot.parsing.lexer.tokens.END* attribute), 305
- type (*robot.parsing.lexer.tokens.EOS* attribute), 303
- type (*robot.parsing.lexer.tokens.Token* attribute), 301
- type (*robot.parsing.model.blocks.If* attribute), 309
- type (*robot.parsing.model.blocks.Try* attribute), 309
- type (*robot.parsing.model.statements.Arguments* attribute), 333
- type (*robot.parsing.model.statements.Break* attribute), 349
- type (*robot.parsing.model.statements.Comment* attribute), 350
- type (*robot.parsing.model.statements.Continue* attribute), 348
- type (*robot.parsing.model.statements.DefaultTags* attribute), 321
- type (*robot.parsing.model.statements.Documentation* attribute), 318
- type (*robot.parsing.model.statements.DocumentationOrMetadata* attribute), 312
- type (*robot.parsing.model.statements.ElseHeader* attribute), 341
- type (*robot.parsing.model.statements.ElseIfHeader* attribute), 340
- type (*robot.parsing.model.statements.EmptyLine* attribute), 352
- type (*robot.parsing.model.statements.End* attribute), 345
- type (*robot.parsing.model.statements.Error* attribute), 351
- type (*robot.parsing.model.statements.ExceptHeader* attribute), 343
- type (*robot.parsing.model.statements.FinallyHeader* attribute), 344
- type (*robot.parsing.model.statements.Fixture* attribute), 315
- type (*robot.parsing.model.statements.ForceTags* attribute), 320
- type (*robot.parsing.model.statements.ForHeader* attribute), 337
- type (*robot.parsing.model.statements.IfElseHeader* attribute), 338
- type (*robot.parsing.model.statements.IfHeader* attribute), 338
- type (*robot.parsing.model.statements.InlineIfHeader* attribute), 339
- type (*robot.parsing.model.statements.KeywordCall* attribute), 335
- type (*robot.parsing.model.statements.KeywordName* attribute), 328
- type (*robot.parsing.model.statements.LibraryImport* attribute), 316
- type (*robot.parsing.model.statements.LoopControl* attribute), 348
- type (*robot.parsing.model.statements.Metadata* attribute), 319
- type (*robot.parsing.model.statements.MultiValue* attribute), 314
- type (*robot.parsing.model.statements.NoArgumentHeader* attribute), 342
- type (*robot.parsing.model.statements.ResourceImport* attribute), 317
- type (*robot.parsing.model.statements.Return* attribute), 334
- type (*robot.parsing.model.statements.ReturnStatement* attribute), 347
- type (*robot.parsing.model.statements.SectionHeader* attribute), 315
- type (*robot.parsing.model.statements.Setup* attribute), 329
- type (*robot.parsing.model.statements.SingleValue* attribute), 313
- type (*robot.parsing.model.statements.Statement* attribute), 311
- type (*robot.parsing.model.statements.SuiteSetup* attribute), 322

- `type (robot.parsing.model.statements.SuiteTeardown attribute), 323`
- `type (robot.parsing.model.statements.Tags attribute), 331`
- `type (robot.parsing.model.statements.Teardown attribute), 330`
- `type (robot.parsing.model.statements.Template attribute), 332`
- `type (robot.parsing.model.statements.TemplateArguments attribute), 336`
- `type (robot.parsing.model.statements.TestCaseName attribute), 328`
- `type (robot.parsing.model.statements.TestSetup attribute), 323`
- `type (robot.parsing.model.statements.TestTeardown attribute), 324`
- `type (robot.parsing.model.statements.TestTemplate attribute), 325`
- `type (robot.parsing.model.statements.TestTimeout attribute), 326`
- `type (robot.parsing.model.statements.Timeout attribute), 333`
- `type (robot.parsing.model.statements.TryHeader attribute), 343`
- `type (robot.parsing.model.statements.Variable attribute), 327`
- `type (robot.parsing.model.statements.VariablesImport attribute), 318`
- `type (robot.parsing.model.statements.WhileHeader attribute), 346`
- `type (robot.result.model.Break attribute), 454`
- `type (robot.result.model.Continue attribute), 451`
- `type (robot.result.model.For attribute), 433`
- `type (robot.result.model.ForIteration attribute), 429`
- `type (robot.result.model.If attribute), 442`
- `type (robot.result.model.IfBranch attribute), 440`
- `type (robot.result.model.Keyword attribute), 457`
- `type (robot.result.model.Message attribute), 428`
- `type (robot.result.model.Return attribute), 449`
- `type (robot.result.model.Try attribute), 447`
- `type (robot.result.model.TryBranch attribute), 444`
- `type (robot.result.model.While attribute), 438`
- `type (robot.result.model.WhileIteration attribute), 433`
- `type (robot.running.arguments.typeconverters.BooleanConverter attribute), 496`
- `type (robot.running.arguments.typeconverters.ByteArrayConverter attribute), 498`
- `type (robot.running.arguments.typeconverters.BytesConverter attribute), 498`
- `type (robot.running.arguments.typeconverters.CombinedConverter attribute), 502`
- `type (robot.running.arguments.typeconverters.CustomConverter attribute), 503`
- `type (robot.running.arguments.typeconverters.DateConverter attribute), 499`
- `type (robot.running.arguments.typeconverters.DateTimeConverter attribute), 499`
- `type (robot.running.arguments.typeconverters.DecimalConverter attribute), 498`
- `type (robot.running.arguments.typeconverters.DictionaryConverter attribute), 501`
- `type (robot.running.arguments.typeconverters.EnumConverter attribute), 495`
- `type (robot.running.arguments.typeconverters.FloatConverter attribute), 497`
- `type (robot.running.arguments.typeconverters.FrozenSetConverter attribute), 502`
- `type (robot.running.arguments.typeconverters.IntegerConverter attribute), 497`
- `type (robot.running.arguments.typeconverters.ListConverter attribute), 500`
- `type (robot.running.arguments.typeconverters.NoneConverter attribute), 500`
- `type (robot.running.arguments.typeconverters.SetConverter attribute), 502`
- `type (robot.running.arguments.typeconverters.StringConverter attribute), 496`
- `type (robot.running.arguments.typeconverters.TimeDeltaConverter attribute), 500`
- `type (robot.running.arguments.typeconverters.TupleConverter attribute), 501`
- `type (robot.running.arguments.typeconverters.TypeConverter attribute), 495`
- `type (robot.running.model.Break attribute), 532`
- `type (robot.running.model.Continue attribute), 531`
- `type (robot.running.model.For attribute), 521`
- `type (robot.running.model.If attribute), 525`
- `type (robot.running.model.IfBranch attribute), 524`
- `type (robot.running.model.Keyword attribute), 520`
- `type (robot.running.model.Return attribute), 529`
- `type (robot.running.model.Try attribute), 528`
- `type (robot.running.model.TryBranch attribute), 527`
- `type (robot.running.model.While attribute), 523`
- `type (robot.running.timeouts.KeywordTimeout attribute), 510`
- `type (robot.running.timeouts.TestTimeout attribute), 510`
- `type_docs (robot.libdocpkg.model.LibraryDoc attribute), 37`
- `converter_name (robot.running.arguments.typeconverters.BooleanConverter attribute), 496`
- `type_name (robot.running.arguments.typeconverters.ByteArrayConverter attribute), 498`
- `converter_name (robot.running.arguments.typeconverters.BytesConverter attribute), 498`
- `converter_name (robot.running.arguments.typeconverters.CombinedConverter attribute), 502`
- `type_name (robot.running.arguments.typeconverters.CustomConverter attribute), 503`
- `type_name (robot.running.arguments.typeconverters.DateConverter attribute), 499`
- `type_name (robot.running.arguments.typeconverters.DateTimeConverter attribute), 499`
- `type_name (robot.running.arguments.typeconverters.DecimalConverter attribute), 498`
- `type_name (robot.running.arguments.typeconverters.DictionaryConverter attribute), 501`
- `type_name (robot.running.arguments.typeconverters.EnumConverter attribute), 495`
- `type_name (robot.running.arguments.typeconverters.FloatConverter attribute), 497`
- `type_name (robot.running.arguments.typeconverters.FrozenSetConverter attribute), 502`
- `type_name (robot.running.arguments.typeconverters.IntegerConverter attribute), 497`
- `type_name (robot.running.arguments.typeconverters.ListConverter attribute), 500`
- `type_name (robot.running.arguments.typeconverters.NoneConverter attribute), 500`
- `type_name (robot.running.arguments.typeconverters.SetConverter attribute), 502`
- `type_name (robot.running.arguments.typeconverters.StringConverter attribute), 496`
- `type_name (robot.running.arguments.typeconverters.TimeDeltaConverter attribute), 500`
- `type_name (robot.running.arguments.typeconverters.TupleConverter attribute), 501`
- `type_name (robot.running.arguments.typeconverters.TypeConverter attribute), 495`
- `type_name (robot.running.model.Break attribute), 532`
- `type_name (robot.running.model.Continue attribute), 531`
- `type_name (robot.running.model.For attribute), 521`
- `type_name (robot.running.model.If attribute), 525`
- `type_name (robot.running.model.IfBranch attribute), 524`
- `type_name (robot.running.model.Keyword attribute), 520`
- `type_name (robot.running.model.Return attribute), 529`
- `type_name (robot.running.model.Try attribute), 528`
- `type_name (robot.running.model.TryBranch attribute), 527`
- `type_name (robot.running.model.While attribute), 523`
- `type_name (robot.running.timeouts.KeywordTimeout attribute), 510`
- `type_name (robot.running.timeouts.TestTimeout attribute), 510`

attribute), 503  
 type\_name (robot.running.arguments.typeconverters.DateConverter method), 131  
 attribute), 499  
 type\_name (robot.running.arguments.typeconverters.DateTimeConverter method), 173  
 attribute), 499  
 type\_name (robot.running.arguments.typeconverters.DecimalConverter method), 187  
 attribute), 498  
 type\_name (robot.running.arguments.typeconverters.DictionaryConverter method), 159  
 attribute), 501  
 type\_name (robot.running.arguments.typeconverters.EnumConverter method), 145  
 attribute), 496  
 type\_name (robot.running.arguments.typeconverters.FloatConverter method), 131  
 attribute), 497  
 type\_name (robot.running.arguments.typeconverters.FrozenSetConverter method), 173  
 attribute), 502  
 type\_name (robot.running.arguments.typeconverters.IntegerConverter method), 187  
 attribute), 497  
 type\_name (robot.running.arguments.typeconverters.ListConverter method), 159  
 attribute), 500  
 type\_name (robot.running.arguments.typeconverters.NoneConverter method), 145  
 attribute), 500  
 type\_name (robot.running.arguments.typeconverters.SetConverter method), 131  
 attribute), 502  
 type\_name (robot.running.arguments.typeconverters.StringConverter method), 173  
 attribute), 496  
 type\_name (robot.running.arguments.typeconverters.TimeDeltaConverter method), 187  
 attribute), 500  
 type\_name (robot.running.arguments.typeconverters.TupleConverter method), 159  
 attribute), 501  
 type\_name (robot.running.arguments.typeconverters.TypeConverter method), 560  
 attribute), 495  
 TypeAndArguments (class in robot.variables.search), 573  
 robot.parsing.lexer.statemlexers), 294  
 TypeConverter (class in robot.running.arguments.typeconverters), 495  
 TYPED\_DICT (robot.libdocpkg.datatypes.TypeDoc attribute), 36  
 TypedDictItem (class in robot.libdocpkg.datatypes), 36  
 TypeDoc (class in robot.libdocpkg.datatypes), 36  
 types (robot.running.arguments.argumentspec.ArgInfo attribute), 494  
 types (robot.running.arguments.argumentspec.ArgumentSpec attribute), 494  
 types\_reprs (robot.running.arguments.argumentspec.ArgInfo attribute), 494  
 TypeValidator (class in robot.running.arguments.typevalidator), 503  
 unbind() (robot.libraries.dialogs\_py.MessageDialog method), 145  
 unbind() (robot.libraries.dialogs\_py.SelectionDialog method), 173  
 unbind() (robot.libraries.dialogs\_py.PassFailDialog method), 187  
 unbind() (robot.libraries.dialogs\_py.SelectionDialog method), 159  
 unbind\_all() (robot.libraries.dialogs\_py.InputDialog method), 145  
 unbind\_all() (robot.libraries.dialogs\_py.MessageDialog method), 173  
 unbind\_all() (robot.libraries.dialogs\_py.SelectionDialog method), 187  
 unbind\_all() (robot.libraries.dialogs\_py.InputDialog method), 159  
 unbind\_all() (robot.libraries.dialogs\_py.MessageDialog method), 145  
 unbind\_all() (robot.libraries.dialogs\_py.SelectionDialog method), 173  
 unbind\_all() (robot.libraries.dialogs\_py.InputDialog method), 187  
 unbind\_all() (robot.libraries.dialogs\_py.SelectionDialog method), 159  
 unescape() (robot.utils.escaping.Unescaper method), 560  
 unescape\_variable\_syntax() (in module robot.variables.search), 573  
 Unescaper (class in robot.utils.escaping), 560  
 unhandled\_tokens (robot.parsing.parser.blockparsers.BlockParser attribute), 353  
 unhandled\_tokens (robot.parsing.parser.blockparsers.ForParser attribute), 354  
 unhandled\_tokens (robot.parsing.parser.blockparsers.IfParser attribute), 354  
 unhandled\_tokens (robot.parsing.parser.blockparsers.KeywordParser attribute), 354  
 unhandled\_tokens (robot.parsing.parser.blockparsers.NestedBlockParser attribute), 354  
 unhandled\_tokens (robot.parsing.parser.blockparsers.TestCaseParser attribute), 354  
 unhandled\_tokens (robot.parsing.parser.blockparsers.TryParser attribute), 354  
 unhandled\_tokens (robot.parsing.parser.blockparsers.WhileParser attribute), 354  
 unregister() (robot.output.listenermethods.LibraryListenerMethods method), 272  
 unregister() (robot.output.listeners.LibraryListeners method), 272  
 unregister\_console\_logger()

## U

[\(robot.output.logger.Logger method\)](#), 273  
[unregister\\_logger\(\)](#) ([robot.output.logger.Logger method](#)), 273  
[unregister\\_xml\\_logger\(\)](#) ([robot.output.logger.Logger method](#)), 273  
[unstrip\(\)](#) ([robot.libraries.XML.NameSpaceStripper method](#)), 122  
[update\(\)](#) ([robot.libraries.dialogs\\_py.InputDialog method](#)), 145  
[update\(\)](#) ([robot.libraries.dialogs\\_py.MessageDialog method](#)), 131  
[update\(\)](#) ([robot.libraries.dialogs\\_py.MultipleSelectionDialog method](#)), 173  
[update\(\)](#) ([robot.libraries.dialogs\\_py.PassFailDialog method](#)), 187  
[update\(\)](#) ([robot.libraries.dialogs\\_py.SelectionDialog method](#)), 159  
[update\(\)](#) ([robot.model.metadata.Metadata method](#)), 230  
[update\(\)](#) ([robot.utils.dotdict.DotDict method](#)), 559  
[update\(\)](#) ([robot.utils.normalizing.NormalizedDict method](#)), 566  
[update\(\)](#) ([robot.variables.scopes.GlobalVariables method](#)), 571  
[update\(\)](#) ([robot.variables.scopes.SetVariables method](#)), 572  
[update\(\)](#) ([robot.variables.store.VariableStore method](#)), 573  
[update\(\)](#) ([robot.variables.variables.Variables method](#)), 574  
[update\\_idletasks\(\)](#) ([robot.libraries.dialogs\\_py.InputDialog method](#)), 145  
[update\\_idletasks\(\)](#) ([robot.libraries.dialogs\\_py.MessageDialog method](#)), 131  
[update\\_idletasks\(\)](#) ([robot.libraries.dialogs\\_py.MultipleSelectionDialog method](#)), 173  
[update\\_idletasks\(\)](#) ([robot.libraries.dialogs\\_py.PassFailDialog method](#)), 187  
[update\\_idletasks\(\)](#) ([robot.libraries.dialogs\\_py.SelectionDialog method](#)), 159  
[usage](#) ([robot.reporting.logreportwriters.LogWriter attribute](#)), 359  
[usage](#) ([robot.reporting.logreportwriters.ReportWriter attribute](#)), 359  
[user\\_agent](#) ([robot.libraries.Remote.TimeoutHTTPTransport attribute](#)), 95  
[user\\_agent](#) ([robot.libraries.Remote.TimeoutHTTPTransport attribute](#)), 95  
[UserErrorHandler](#) (class in [robot.running.usererrorhandler](#)), 551  
[UserKeyword](#) (class in [robot.running.model](#)), 538  
[UserKeywordArgumentParser](#) (class in [robot.running.arguments.argumentparser](#)), 493  
[UserKeywordExecutionFailed](#), 577  
[UserKeywordHandler](#) (class in [robot.running.userkeyword](#)), 552  
[UserKeywordRunner](#) (class in [robot.running.userkeywordrunner](#)), 552  
[UserLibrary](#) (class in [robot.running.userkeyword](#)), 552

## V

[validate\(\)](#) ([robot.libdoc.LibDoc method](#)), 580  
[validate\(\)](#) ([robot.parsing.model.blocks.Block method](#)), 306  
[validate\(\)](#) ([robot.parsing.model.blocks.CommentSection method](#)), 308  
[validate\(\)](#) ([robot.parsing.model.blocks.File method](#)), 306  
[validate\(\)](#) ([robot.parsing.model.blocks.For method](#)), 309  
[validate\(\)](#) ([robot.parsing.model.blocks.HeaderAndBody method](#)), 306  
[validate\(\)](#) ([robot.parsing.model.blocks.If method](#)), 309  
[validate\(\)](#) ([robot.parsing.model.blocks.Keyword method](#)), 308  
[validate\(\)](#) ([robot.parsing.model.blocks.KeywordSection method](#)), 308  
[validate\(\)](#) ([robot.parsing.model.blocks.Section method](#)), 307  
[validate\(\)](#) ([robot.parsing.model.blocks.SettingSection method](#)), 307  
[validate\(\)](#) ([robot.parsing.model.blocks.TestCase method](#)), 308  
[validate\(\)](#) ([robot.parsing.model.blocks.TestCaseSection method](#)), 307  
[validate\(\)](#) ([robot.parsing.model.blocks.Try method](#)), 309  
[validate\(\)](#) ([robot.parsing.model.blocks.VariableSection method](#)), 307  
[validate\(\)](#) ([robot.parsing.model.blocks.While method](#)), 310  
[validate\(\)](#) ([robot.parsing.model.statements.Arguments method](#)), 334  
[validate\(\)](#) ([robot.parsing.model.statements.Break method](#)), 350  
[validate\(\)](#) ([robot.parsing.model.statements.Comment method](#)), 351  
[validate\(\)](#) ([robot.parsing.model.statements.Continue method](#)), 349



`validate()` (`robot.parsing.model.statements.DefaultTags` method), 322  
`validate()` (`robot.parsing.model.statements.Documentation` method), 319  
`validate()` (`robot.parsing.model.statements.DocumentationMetadata` method), 312  
`validate()` (`robot.parsing.model.statements.ElseHeader` method), 341  
`validate()` (`robot.parsing.model.statements.ElseIfHeader` method), 341  
`validate()` (`robot.parsing.model.statements.EmptyLine` method), 352  
`validate()` (`robot.parsing.model.statements.End` method), 346  
`validate()` (`robot.parsing.model.statements.Error` method), 352  
`validate()` (`robot.parsing.model.statements.ExceptHeader` method), 344  
`validate()` (`robot.parsing.model.statements.FinallyHeader` method), 345  
`validate()` (`robot.parsing.model.statements.Fixture` method), 315  
`validate()` (`robot.parsing.model.statements.ForceTags` method), 321  
`validate()` (`robot.parsing.model.statements.ForHeader` method), 337  
`validate()` (`robot.parsing.model.statements.IfElseHeader` method), 338  
`validate()` (`robot.parsing.model.statements.IfHeader` method), 339  
`validate()` (`robot.parsing.model.statements.InlineIfHeader` method), 340  
`validate()` (`robot.parsing.model.statements.KeywordCall` method), 336  
`validate()` (`robot.parsing.model.statements.KeywordName` method), 329  
`validate()` (`robot.parsing.model.statements.LibraryImport` method), 317  
`validate()` (`robot.parsing.model.statements.LoopControl` method), 348  
`validate()` (`robot.parsing.model.statements.Metadata` method), 320  
`validate()` (`robot.parsing.model.statements.MultiValue` method), 314  
`validate()` (`robot.parsing.model.statements.NoArgumentHeader` method), 342  
`validate()` (`robot.parsing.model.statements.ResourceImport` method), 318  
`validate()` (`robot.parsing.model.statements.Return` method), 335  
`validate()` (`robot.parsing.model.statements.ReturnStatement` method), 347  
`validate()` (`robot.parsing.model.statements.SectionHeader` method), 316  
`validate()` (`robot.parsing.model.statements.Setup` method), 330  
`validate()` (`robot.parsing.model.statements.SingleValue` method), 313  
`validate()` (`robot.parsing.model.statements.Statement` method), 312  
`validate()` (`robot.parsing.model.statements.SuiteSetup` method), 323  
`validate()` (`robot.parsing.model.statements.SuiteTeardown` method), 323  
`validate()` (`robot.parsing.model.statements.Tags` method), 332  
`validate()` (`robot.parsing.model.statements.Tearardown` method), 331  
`validate()` (`robot.parsing.model.statements.Template` method), 332  
`validate()` (`robot.parsing.model.statements.TemplateArguments` method), 337  
`validate()` (`robot.parsing.model.statements.TestCaseName` method), 328  
`validate()` (`robot.parsing.model.statements.TestSetup` method), 324  
`validate()` (`robot.parsing.model.statements.TestTeardown` method), 325  
`validate()` (`robot.parsing.model.statements.TestTemplate` method), 326  
`validate()` (`robot.parsing.model.statements.TestTimeout` method), 327  
`validate()` (`robot.parsing.model.statements.Timeout` method), 333  
`validate()` (`robot.parsing.model.statements.TryHeader` method), 343  
`validate()` (`robot.parsing.model.statements.Variable` method), 327  
`validate()` (`robot.parsing.model.statements.VariablesImport` method), 318  
`validate()` (`robot.parsing.model.statements.WhileHeader` method), 346  
`validate()` (`robot.rebot.Rebot` method), 582  
`validate()` (`robot.run.RobotFramework` method), 583  
`validate()` (`robot.running.arguments.argumentvalidator.ArgumentValidator` method), 494  
`validate()` (`robot.running.arguments.typevalidator.TypeValidator` method), 503  
`validate()` (`robot.testdoc.TestDoc` method), 585  
`validate()` (`robot.utils.application.Application` method), 553  
`validate()` (`robot.variables.assigner.AssignmentValidator` method), 568  
`validate_assignment()` (`robot.variables.assigner.VariableAssignment` method), 568  
`validate_command()` (`robot.libdocpkg.consoleviewer.ConsoleViewer` method), 568

class method), 36  
 validate\_flatten\_keyword() (in module `robot.result.flattenkeywordmatcher`), 379  
 validate\_model() (robot.parsing.model.blocks.Block value attribute), 306  
 validate\_model() (robot.parsing.model.blocks.Comment value attribute), 308  
 validate\_model() (robot.parsing.model.blocks.File value attribute), 306  
 validate\_model() (robot.parsing.model.blocks.For value attribute), 309  
 validate\_model() (robot.parsing.model.blocks.HeaderAndBody value attribute), 306  
 validate\_model() (robot.parsing.model.blocks.If value attribute), 309  
 validate\_model() (robot.parsing.model.blocks.Keyword value attribute), 308  
 validate\_model() (robot.parsing.model.blocks.KeywordList value attribute), 308  
 validate\_model() (robot.parsing.model.blocks.Section value attribute), 307  
 validate\_model() (robot.parsing.model.blocks.Setting value attribute), 307  
 validate\_model() (robot.parsing.model.blocks.TestCase value attribute), 308  
 validate\_model() (robot.parsing.model.blocks.TestCase value attribute), 307  
 validate\_model() (robot.parsing.model.blocks.Try value attribute), 310  
 validate\_model() (robot.parsing.model.blocks.Variable value attribute), 307  
 validate\_model() (robot.parsing.model.blocks.While value attribute), 310  
 validate\_type\_dict() (robot.running.arguments.typevalidator.TypeValidator value attribute), 503  
 ValidationContext (class in `robot.parsing.model.blocks`), 310  
 value (robot.parsing.lexer.tokens.END attribute), 305  
 value (robot.parsing.lexer.tokens.EOS attribute), 303  
 value (robot.parsing.lexer.tokens.Token attribute), 301  
 value (robot.parsing.model.statements.Documentation attribute), 319  
 value (robot.parsing.model.statements.Metadata attribute), 320  
 value (robot.parsing.model.statements.SingleValue attribute), 313  
 value (robot.parsing.model.statements.Template attribute), 332  
 value (robot.parsing.model.statements.TestTemplate attribute), 326  
 value (robot.parsing.model.statements.TestTimeout attribute), 327  
 value (robot.parsing.model.statements.Timeout attribute), 333  
 value (robot.parsing.model.statements.Variable attribute), 327  
 value\_types (robot.running.arguments.typeconverters.BooleanConverter attribute), 496  
 value\_types (robot.running.arguments.typeconverters.ByteArrayConverter attribute), 498  
 value\_types (robot.running.arguments.typeconverters.BytesConverter attribute), 498  
 value\_types (robot.running.arguments.typeconverters.CombinedConverter attribute), 503  
 value\_types (robot.running.arguments.typeconverters.CustomConverter attribute), 503  
 value\_types (robot.running.arguments.typeconverters.DateConverter attribute), 499  
 value\_types (robot.running.arguments.typeconverters.DateTimeConverter attribute), 499  
 value\_types (robot.running.arguments.typeconverters.DecimalConverter attribute), 498  
 value\_types (robot.running.arguments.typeconverters.DictionaryConverter attribute), 501  
 value\_types (robot.running.arguments.typeconverters.EnumConverter attribute), 496  
 value\_types (robot.running.arguments.typeconverters.FloatConverter attribute), 497  
 value\_types (robot.running.arguments.typeconverters.FrozenSetConverter attribute), 502  
 value\_types (robot.running.arguments.typeconverters.IntegerConverter attribute), 497  
 value\_types (robot.running.arguments.typeconverters.ListConverter attribute), 500  
 value\_types (robot.running.arguments.typeconverters.NoneConverter attribute), 500  
 value\_types (robot.running.arguments.typeconverters.SetConverter attribute), 502  
 value\_types (robot.running.arguments.typeconverters.StringConverter attribute), 496  
 value\_types (robot.running.arguments.typeconverters.TimeDeltaConverter attribute), 500  
 value\_types (robot.running.arguments.typeconverters.TupleConverter attribute), 501  
 value\_types (robot.running.arguments.typeconverters.TypeConverter attribute), 495  
 ValueHandler (class in `robot.result.xmllelementhandlers`), 490  
 values (robot.model.control.For attribute), 204  
 values (robot.model.control.Return attribute), 211  
 values (robot.parsing.model.blocks.For attribute), 309  
 values (robot.parsing.model.statements.Arguments attribute), 334  
 values (robot.parsing.model.statements.Break attribute), 350  
 values (robot.parsing.model.statements.Continue attribute), 349

values (robot.parsing.model.statements.DefaultTags attribute), 322	variable (robot.parsing.model.blocks.Try attribute), 309
values (robot.parsing.model.statements.End attribute), 346	variable (robot.parsing.model.statements.ExceptHeader attribute), 344
values (robot.parsing.model.statements.FinallyHeader attribute), 345	variable (robot.result.model.TryBranch attribute), 444
values (robot.parsing.model.statements.ForceTags attribute), 321	variable (robot.running.model.TryBranch attribute), 527
values (robot.parsing.model.statements.ForHeader attribute), 337	variable() (robot.parsing.lexer.sections.InitFileSections method), 292
values (robot.parsing.model.statements.LoopControl attribute), 348	variable() (robot.parsing.lexer.sections.ResourceFileSections method), 291
values (robot.parsing.model.statements.MultiValue attribute), 313	variable() (robot.parsing.lexer.sections.Sections method), 291
values (robot.parsing.model.statements.NoArgumentHeader attribute), 342	variable() (robot.parsing.lexer.sections.TestCaseFileSections method), 291
values (robot.parsing.model.statements.Return attribute), 335	variable_files (robot.conf.settings.RobotSettings attribute), 32
values (robot.parsing.model.statements.ReturnStatement attribute), 347	VARIABLE_HEADER (robot.parsing.lexer.tokens.END attribute), 305
values (robot.parsing.model.statements.Tags attribute), 332	VARIABLE_HEADER (robot.parsing.lexer.tokens.EOS attribute), 303
values (robot.parsing.model.statements.TryHeader attribute), 343	VARIABLE_HEADER (robot.parsing.lexer.tokens.Token attribute), 299
values (robot.result.model.For attribute), 433	variable_markers (robot.parsing.lexer.sections.InitFileSections attribute), 292
values (robot.result.model.Return attribute), 449	variable_markers (robot.parsing.lexer.sections.ResourceFileSections attribute), 291
values (robot.running.model.For attribute), 521	variable_markers (robot.parsing.lexer.sections.Sections attribute), 290
values (robot.running.model.Return attribute), 530	variable_markers (robot.parsing.lexer.sections.TestCaseFileSections attribute), 291
values() (robot.model.metadata.Metadata method), 230	variable_not_found() (in module robot.variables.notfound), 570
values() (robot.running.importer.ImportCache method), 514	variable_section() (robot.parsing.lexer.context.FileContext method), 288
values() (robot.utils.dotdict.DotDict method), 559	variable_section() (robot.parsing.lexer.context.InitFileContext method), 289
values() (robot.utils.normalizing.NormalizedDict method), 566	variable_section() (robot.parsing.lexer.context.ResourceFileContext method), 289
values() (robot.variables.evaluation.EvaluationNamespace method), 568	variable_section() (robot.parsing.lexer.context.TestCaseFileContext method), 289
VAR_NAMED (robot.running.arguments.argumentspec.ArgInfo attribute), 494	variable_should_exist() (robot.libraries.BuiltIn.BuiltIn method), 64
VAR_POSITIONAL (robot.running.arguments.argumentspec.ArgInfo method), 494	variable_should_not_exist() (robot.libraries.BuiltIn.BuiltIn method), 64
VarHandler (class in robot.result.xmlélémenthandlers), 489	variable_state() (robot.variables.search.VariableSearcher method), 573
Variable (class in robot.parsing.model.statements), 327	
Variable (class in robot.running.model), 538	
variable (robot.model.control.TryBranch attribute), 209	
VARIABLE (robot.parsing.lexer.tokens.END attribute), 305	
VARIABLE (robot.parsing.lexer.tokens.EOS attribute), 303	
VARIABLE (robot.parsing.lexer.tokens.Token attribute), 300	

VariableAssigner	(class in <i>robot.variables.assigner</i> ), 568	in <i>robot.parsing.model.statements</i> ), 318
VariableAssignment	(class in <i>robot.variables.assigner</i> ), 568	VariableStore (class in <i>robot.variables.store</i> ), 573
VariableError	575	VariableTableSetter (class in <i>robot.variables.tablesetter</i> ), 573
VariableFileSetter	(class in <i>robot.variables.filesetter</i> ), 569	VariableTableValue() (in module <i>robot.variables.tablesetter</i> ), 573
VariableFinder	(class in <i>robot.variables.finders</i> ), 569	VariableTableValueBase (class in <i>robot.variables.tablesetter</i> ), 573
VariableIterator	(class in <i>robot.variables.search</i> ), 573	VerboseOutput (class in <i>robot.output.console.verbose</i> ), 269
VariableLexer	(class in <i>robot.parsing.lexer.statementlexers</i> ), 296	VerboseWriter (class in <i>robot.output.console.verbose</i> ), 269
VariableMatch	(class in <i>robot.variables.search</i> ), 572	version() ( <i>robot.libdocpkg.consoleviewer.ConsoleViewer</i> method), 36
VariableReplacer	(class in <i>robot.running.arguments.argumentresolver</i> ), 493	view() ( <i>robot.libdocpkg.consoleviewer.ConsoleViewer</i> method), 36
VariableReplacer	(class in <i>robot.variables.replacer</i> ), 570	visit() ( <i>robot.model.body.BaseBody</i> method), 195
Variables	(class in <i>robot.variables.variables</i> ), 574	visit() ( <i>robot.model.body.Body</i> method), 197
variables	( <i>robot.conf.settings.RobotSettings</i> attribute), 32	visit() ( <i>robot.model.body.Branches</i> method), 199
variables	( <i>robot.model.control.For</i> attribute), 203	visit() ( <i>robot.model.control.Break</i> method), 215
VARIABLES	( <i>robot.parsing.lexer.tokens.END</i> attribute), 305	visit() ( <i>robot.model.control.Continue</i> method), 214
VARIABLES	( <i>robot.parsing.lexer.tokens.EOS</i> attribute), 303	visit() ( <i>robot.model.control.For</i> method), 204
VARIABLES	( <i>robot.parsing.lexer.tokens.Token</i> attribute), 300	visit() ( <i>robot.model.control.If</i> method), 208
variables	( <i>robot.parsing.model.blocks.For</i> attribute), 309	visit() ( <i>robot.model.control.IfBranch</i> method), 206
variables	( <i>robot.parsing.model.statements.ForHeader</i> attribute), 337	visit() ( <i>robot.model.control.Return</i> method), 212
variables	( <i>robot.result.model.For</i> attribute), 433	visit() ( <i>robot.model.control.Try</i> method), 210
variables	( <i>robot.result.model.ForIteration</i> attribute), 429	visit() ( <i>robot.model.control.TryBranch</i> method), 209
variables	( <i>robot.running.model.For</i> attribute), 521	visit() ( <i>robot.model.control.While</i> method), 205
variables	( <i>robot.running.model.ResourceFile</i> attribute), 538	visit() ( <i>robot.model.itemlist.ItemList</i> method), 224
variables()	( <i>robot.running.model.Imports</i> method), 539	visit() ( <i>robot.model.keyword.Keyword</i> method), 225
VariableScopes	(class in <i>robot.variables.scopes</i> ), 570	visit() ( <i>robot.model.keyword.Keywords</i> method), 227
VariableSearcher	(class in <i>robot.variables.search</i> ), 572	visit() ( <i>robot.model.message.Message</i> method), 228
VariableSection	(class in <i>robot.parsing.model.blocks</i> ), 307	visit() ( <i>robot.model.message.Messages</i> method), 229
VariableSectionHeaderLexer	(class in <i>robot.parsing.lexer.statementlexers</i> ), 294	visit() ( <i>robot.model.statistics.Statistics</i> method), 235
VariableSectionLexer	(class in <i>robot.parsing.lexer.blocklexers</i> ), 284	visit() ( <i>robot.model.stats.CombinedTagStat</i> method), 241
VariableSectionParser	(class in <i>robot.parsing.parser.fileparser</i> ), 355	visit() ( <i>robot.model.stats.Stat</i> method), 240
VariablesImport	(class in <i>robot.parsing.model.statements</i> ), 318	visit() ( <i>robot.model.stats.SuiteStat</i> method), 240
		visit() ( <i>robot.model.stats.TagStat</i> method), 241
		visit() ( <i>robot.model.stats.TotalStat</i> method), 240
		visit() ( <i>robot.model.sitestatistics.SuiteStatistics</i> method), 241
		visit() ( <i>robot.model.tagstatistics.TagStatistics</i> method), 247
		visit() ( <i>robot.model.testcase.TestCase</i> method), 249
		visit() ( <i>robot.model.testcase.TestCases</i> method), 250
		visit() ( <i>robot.model.testsuite.TestSuite</i> method), 252
		visit() ( <i>robot.model.testsuite.TestSuites</i> method), 253
		visit() ( <i>robot.model.totalstatistics.TotalStatistics</i> method), 253
		visit() ( <i>robot.output.loggerhelper.Message</i> method), 276
		visit() ( <i>robot.parsing.model.blocks.FirstStatementFinder</i> method), 311



- `visit()` (*robot.parsing.model.blocks.LastStatementFinder* method), 311
- `visit()` (*robot.parsing.model.blocks.ModelValidator* method), 310
- `visit()` (*robot.parsing.model.blocks.ModelWriter* method), 310
- `visit()` (*robot.parsing.model.visitor.ModelTransformer* method), 353
- `visit()` (*robot.parsing.model.visitor.ModelVisitor* method), 353
- `visit()` (*robot.parsing.suitestructure.SuiteStructure* method), 356
- `visit()` (*robot.result.executionerrors.ExecutionErrors* method), 377
- `visit()` (*robot.result.executionresult.CombinedResult* method), 379
- `visit()` (*robot.result.executionresult.Result* method), 378
- `visit()` (*robot.result.model.Body* method), 423
- `visit()` (*robot.result.model.Branches* method), 425
- `visit()` (*robot.result.model.Break* method), 454
- `visit()` (*robot.result.model.Continue* method), 451
- `visit()` (*robot.result.model.For* method), 433
- `visit()` (*robot.result.model.ForIteration* method), 429
- `visit()` (*robot.result.model.If* method), 442
- `visit()` (*robot.result.model.IfBranch* method), 440
- `visit()` (*robot.result.model.Iterations* method), 427
- `visit()` (*robot.result.model.Keyword* method), 457
- `visit()` (*robot.result.model.Message* method), 428
- `visit()` (*robot.result.model.Return* method), 449
- `visit()` (*robot.result.model.TestCase* method), 459
- `visit()` (*robot.result.model.TestSuite* method), 462
- `visit()` (*robot.result.model.Try* method), 447
- `visit()` (*robot.result.model.TryBranch* method), 444
- `visit()` (*robot.result.model.While* method), 438
- `visit()` (*robot.result.model.WhileIteration* method), 433
- `visit()` (*robot.running.builder.parsers.ErrorReporter* method), 505
- `visit()` (*robot.running.builder.transformers.ForBuilder* method), 509
- `visit()` (*robot.running.builder.transformers.IfBuilder* method), 509
- `visit()` (*robot.running.builder.transformers.KeywordBuilder* method), 508
- `visit()` (*robot.running.builder.transformers.ResourceBuilder* method), 507
- `visit()` (*robot.running.builder.transformers.SettingsBuilder* method), 506
- `visit()` (*robot.running.builder.transformers.SuiteBuilder* method), 507
- `visit()` (*robot.running.builder.transformers.TestCaseBuilder* method), 508
- `visit()` (*robot.running.builder.transformers.TryBuilder* method), 509
- `visit()` (*robot.running.builder.transformers.WhileBuilder* method), 510
- `visit()` (*robot.running.model.Body* method), 517
- `visit()` (*robot.running.model.Break* method), 532
- `visit()` (*robot.running.model.Continue* method), 531
- `visit()` (*robot.running.model.For* method), 521
- `visit()` (*robot.running.model.If* method), 525
- `visit()` (*robot.running.model.IfBranch* method), 524
- `visit()` (*robot.running.model.Imports* method), 539
- `visit()` (*robot.running.model.Keyword* method), 520
- `visit()` (*robot.running.model.Return* method), 530
- `visit()` (*robot.running.model.TestCase* method), 534
- `visit()` (*robot.running.model.TestSuite* method), 538
- `visit()` (*robot.running.model.Try* method), 528
- `visit()` (*robot.running.model.TryBranch* method), 527
- `visit()` (*robot.running.model.While* method), 523
- `visit_arguments()` (*robot.running.builder.transformers.KeywordBuilder* method), 508
- `visit_block()` (*robot.parsing.model.blocks.ModelValidator* method), 310
- `visit_break()` (*robot.conf.gatherfailed.GatherFailedSuites* method), 30
- `visit_break()` (*robot.conf.gatherfailed.GatherFailedTests* method), 26
- `visit_break()` (*robot.model.configurer.SuiteConfigurer* method), 202
- `visit_break()` (*robot.model.filter.EmptySuiteRemover* method), 218
- `visit_break()` (*robot.model.filter.Filter* method), 222
- `visit_break()` (*robot.model.modifier.ModelModifier* method), 233
- `visit_break()` (*robot.model.statistics.StatisticsBuilder* method), 238
- `visit_break()` (*robot.model.tagsetter.TagSetter* method), 245
- `visit_break()` (*robot.model.totalstatistics.TotalStatisticsBuilder* method), 256
- `visit_break()` (*robot.model.visitor.SuiteVisitor* method), 262
- `visit_break()` (*robot.output.console.dotted.StatusReporter* method), 267
- `visit_break()` (*robot.output.xmllogger.XmlLogger* method), 281
- `visit_break()` (*robot.reporting.outputwriter.OutputWriter* method), 363
- `visit_break()` (*robot.reporting.xunitwriter.XUnitFileWriter* method), 370
- `visit_break()` (*robot.result.configurer.SuiteConfigurer* method), 375
- `visit_break()` (*robot.result.keywordremover.AllKeywordsRemover* method), 382

`visit_break()` (`robot.result.keywordremover.ByNameKeywordRemover` `method`), 390  
`visit_continue()` (`robot.model.statistics.StatisticsBuilder` `method`), 238  
`visit_break()` (`robot.result.keywordremover.ByTagKeywordRemover` `method`), 395  
`visit_continue()` (`robot.model.tagsetter.TagSetter` `method`), 245  
`visit_break()` (`robot.result.keywordremover.ForLoopItemsRemover` `method`), 399  
`visit_continue()` (`robot.model.totalstatistics.TotalStatisticsBuilder` `method`), 256  
`visit_break()` (`robot.result.keywordremover.PassedKeywordRemover` `method`), 386  
`visit_continue()` (`robot.model.visitor.SuiteVisitor` `method`), 262  
`visit_break()` (`robot.result.keywordremover.WaitUntilKeywordSucceedsRemover` `method`), 407  
`visit_continue()` (`robot.output.console.dotted.StatusReporter` `method`), 267  
`visit_break()` (`robot.result.keywordremover.WarningAndErrorFinder` `method`), 412  
`visit_continue()` (`robot.output.xmllogger.XmlLogger` `method`), 281  
`visit_break()` (`robot.result.keywordremover.WhileLoopItemsRemover` `method`), 403  
`visit_continue()` (`robot.reporting.outputwriter.OutputWriter` `method`), 363  
`visit_break()` (`robot.result.merger.Merger` `method`), 416  
`visit_continue()` (`robot.reporting.xunitwriter.XUnitFileWriter` `method`), 370  
`visit_break()` (`robot.result.messagefilter.MessageFilter` `method`), 420  
`visit_continue()` (`robot.result.configurer.SuiteConfigurer` `method`), 375  
`visit_break()` (`robot.result.resultbuilder.RemoveKeywords` `method`), 468  
`visit_continue()` (`robot.result.keywordremover.AllKeywordsRemover` `method`), 382  
`visit_break()` (`robot.result.suiteteardownfailed.SuiteTeardownFailed` `method`), 476  
`visit_continue()` (`robot.result.keywordremover.ByNameKeywordRemover` `method`), 390  
`visit_break()` (`robot.result.suiteteardownfailed.SuiteTeardownFailedHandler` `method`), 472  
`visit_continue()` (`robot.result.keywordremover.ByTagKeywordRemover` `method`), 395  
`visit_break()` (`robot.result.visitor.ResultVisitor` `method`), 481  
`visit_continue()` (`robot.result.keywordremover.ForLoopItemsRemover` `method`), 399  
`visit_Break()` (`robot.running.builder.transformers.ForBuilder` `method`), 508  
`visit_continue()` (`robot.result.keywordremover.PassedKeywordRemover` `method`), 387  
`visit_Break()` (`robot.running.builder.transformers.IfBuilder` `method`), 509  
`visit_continue()` (`robot.result.keywordremover.WaitUntilKeywordSucceedsRemover` `method`), 407  
`visit_Break()` (`robot.running.builder.transformers.KeywordBuilder` `method`), 508  
`visit_continue()` (`robot.result.keywordremover.WarningAndErrorFinder` `method`), 412  
`visit_Break()` (`robot.running.builder.transformers.TestCaseBuilder` `method`), 507  
`visit_continue()` (`robot.result.keywordremover.WhileLoopItemsRemover` `method`), 403  
`visit_Break()` (`robot.running.builder.transformers.TryBuilder` `method`), 509  
`visit_continue()` (`robot.result.merger.Merger` `method`), 416  
`visit_Break()` (`robot.running.builder.transformers.WhileBuilder` `method`), 510  
`visit_continue()` (`robot.result.messagefilter.MessageFilter` `method`), 420  
`visit_break()` (`robot.running.randomizer.Randomizer` `method`), 544  
`visit_continue()` (`robot.result.resultbuilder.RemoveKeywords` `method`), 468  
`visit_break()` (`robot.running.suiterunner.SuiteRunner` `method`), 550  
`visit_continue()` (`robot.result.suiteteardownfailed.SuiteTeardownFailed` `method`), 476  
`visit_continue()` (`robot.conf.gatherfailed.GatherFailedSuites` `method`), 30  
`visit_continue()` (`robot.result.suiteteardownfailed.SuiteTeardownFailedHandler` `method`), 472  
`visit_continue()` (`robot.conf.gatherfailed.GatherFailedTests` `method`), 26  
`visit_continue()` (`robot.result.visitor.ResultVisitor` `method`), 481  
`visit_continue()` (`robot.model.configurer.SuiteConfigurer` `method`), 202  
`visit_continue()` (`robot.running.builder.transformers.ForBuilder` `method`), 508  
`visit_continue()` (`robot.model.filter.EmptySuiteRemover` `method`), 218  
`visit_continue()` (`robot.running.builder.transformers.IfBuilder` `method`), 509  
`visit_continue()` (`robot.model.filter.Filter` `method`), 222  
`visit_continue()` (`robot.running.builder.transformers.KeywordBuilder` `method`), 508  
`visit_continue()` (`robot.model.modifier.ModelModifier` `method`), 233  
`visit_continue()` (`robot.running.builder.transformers.TestCaseBuilder` `method`), 507

[visit\\_Continue\(\) \(robot.running.builder.transformers.TryBuilder method\), 509](#)  
[visit\\_Continue\(\) \(robot.running.builder.transformers.WhileBuilder method\), 510](#)  
[visit\\_continue\(\) \(robot.running.randomizer.Randomizer method\), 544](#)  
[visit\\_continue\(\) \(robot.running.suiterunner.SuiteRunner method\), 550](#)  
[visit\\_DefaultTags\(\) \(robot.running.builder.transformers.SettingsBuilder method\), 506](#)  
[visit\\_directory\(\) \(robot.parsing.suitestructure.SuiteStructureVisitor method\), 357](#)  
[visit\\_directory\(\) \(robot.running.builder.builders.SuiteStructureParser method\), 504](#)  
[visit\\_Documentation\(\) \(robot.running.builder.transformers.KeywordBuilder method\), 508](#)  
[visit\\_Documentation\(\) \(robot.running.builder.transformers.ResourceBuilder method\), 507](#)  
[visit\\_Documentation\(\) \(robot.running.builder.transformers.SettingsBuilder method\), 506](#)  
[visit\\_Documentation\(\) \(robot.running.builder.transformers.TestCaseBuilder method\), 507](#)  
[visit\\_Error\(\) \(robot.running.builder.parsers.ErrorReporter method\), 505](#)  
[visit\\_errors\(\) \(robot.output.xmllogger.XmlLogger method\), 282](#)  
[visit\\_errors\(\) \(robot.reporting.outputwriter.OutputWriter method\), 363](#)  
[visit\\_errors\(\) \(robot.reporting.xunitwriter.XUnitFileWriter method\), 366](#)  
[visit\\_errors\(\) \(robot.result.visitor.ResultVisitor method\), 478](#)  
[visit\\_file\(\) \(robot.parsing.suitestructure.SuiteStructureVisitor method\), 357](#)  
[visit\\_file\(\) \(robot.running.builder.builders.SuiteStructureParser method\), 504](#)  
[visit\\_for\(\) \(robot.conf.gatherfailed.GatherFailedSuites method\), 30](#)  
[visit\\_for\(\) \(robot.conf.gatherfailed.GatherFailedTests method\), 26](#)  
[visit\\_for\(\) \(robot.model.configurer.SuiteConfigurer method\), 202](#)  
[visit\\_for\(\) \(robot.model.filter.EmptySuiteRemover method\), 218](#)  
[visit\\_for\(\) \(robot.model.filter.Filter method\), 222](#)  
[visit\\_for\(\) \(robot.model.modifier.ModelModifier method\), 233](#)  
[visit\\_For\(\) \(robot.model.statistics.StatisticsBuilder method\), 238](#)  
[visit\\_For\(\) \(robot.model.tagsetter.TagSetter method\), 245](#)  
[visit\\_for\(\) \(robot.model.totalstatistics.TotalStatisticsBuilder method\), 256](#)  
[visit\\_for\(\) \(robot.model.visitor.SuiteVisitor method\), 260](#)  
[visit\\_for\(\) \(robot.output.console.dotted.StatusReporter method\), 267](#)  
[visit\\_for\(\) \(robot.output.xmllogger.XmlLogger method\), 282](#)  
[visit\\_for\(\) \(robot.reporting.outputwriter.OutputWriter method\), 363](#)  
[visit\\_for\(\) \(robot.reporting.xunitwriter.XUnitFileWriter method\), 370](#)  
[visit\\_for\(\) \(robot.result.configurer.SuiteConfigurer method\), 375](#)  
[visit\\_for\(\) \(robot.result.keywordremover.AllKeywordsRemover method\), 379](#)  
[visit\\_for\(\) \(robot.result.keywordremover.ByNameKeywordRemover method\), 391](#)  
[visit\\_for\(\) \(robot.result.keywordremover.ByTagKeywordRemover method\), 395](#)  
[visit\\_for\(\) \(robot.result.keywordremover.ForLoopItemsRemover method\), 399](#)  
[visit\\_for\(\) \(robot.result.keywordremover.PassedKeywordRemover method\), 387](#)  
[visit\\_for\(\) \(robot.result.keywordremover.WaitUntilKeywordSucceeds method\), 407](#)  
[visit\\_for\(\) \(robot.result.keywordremover.WarningAndErrorFinder method\), 412](#)  
[visit\\_for\(\) \(robot.result.keywordremover.WhileLoopItemsRemover method\), 403](#)  
[visit\\_for\(\) \(robot.result.merger.Merger method\), 416](#)  
[visit\\_for\(\) \(robot.result.messagefilter.MessageFilter method\), 420](#)  
[visit\\_for\(\) \(robot.result.resultbuilder.RemoveKeywords method\), 468](#)  
[visit\\_for\(\) \(robot.result.suitetardownfailed.SuiteTeardownFailed method\), 476](#)  
[visit\\_for\(\) \(robot.result.suitetardownfailed.SuiteTeardownFailureHandler method\), 472](#)  
[visit\\_for\(\) \(robot.result.visitor.ResultVisitor method\), 481](#)  
[visit\\_For\(\) \(robot.running.builder.transformers.ForBuilder method\), 508](#)  
[visit\\_For\(\) \(robot.running.builder.transformers.IfBuilder method\), 509](#)  
[visit\\_For\(\) \(robot.running.builder.transformers.KeywordBuilder method\), 508](#)  
[visit\\_For\(\) \(robot.running.builder.transformers.TestCaseBuilder method\), 507](#)

`visit_For()` (`robot.running.builder.transformers.TryBuilder` method), 382  
method), 509  
`visit_For()` (`robot.running.builder.transformers.WhileBuilder` method), 510  
`visit_for()` (`robot.running.randomizer.Randomizer` method), 544  
`visit_for()` (`robot.running.suiterunner.SuiteRunner` method), 550  
`visit_for_iteration()`  
(`robot.conf.gatherfailed.GatherFailedSuites` method), 30  
`visit_for_iteration()`  
(`robot.conf.gatherfailed.GatherFailedTests` method), 26  
`visit_for_iteration()`  
(`robot.model.configurer.SuiteConfigurer` method), 202  
`visit_for_iteration()`  
(`robot.model.filter.EmptySuiteRemover` method), 218  
`visit_for_iteration()` (`robot.model.filter.Filter` method), 222  
`visit_for_iteration()`  
(`robot.model.modifier.ModelModifier` method), 233  
`visit_for_iteration()`  
(`robot.model.statistics.StatisticsBuilder` method), 238  
`visit_for_iteration()`  
(`robot.model.tagsetter.TagSetter` method), 245  
`visit_for_iteration()`  
(`robot.model.totalstatistics.TotalStatisticsBuilder` method), 257  
`visit_for_iteration()`  
(`robot.model.visitor.SuiteVisitor` method), 260  
`visit_for_iteration()`  
(`robot.output.console.dotted.StatusReporter` method), 267  
`visit_for_iteration()`  
(`robot.output.xmllogger.XmlLogger` method), 282  
`visit_for_iteration()`  
(`robot.reporting.outputwriter.OutputWriter` method), 363  
`visit_for_iteration()`  
(`robot.reporting.xunitwriter.XUnitFileWriter` method), 370  
`visit_for_iteration()`  
(`robot.result.configurer.SuiteConfigurer` method), 375  
`visit_for_iteration()`  
(`robot.result.keywordremover.AllKeywordsRemover` method), 382  
`visit_for_iteration()`  
(`robot.result.keywordremover.ByTagKeywordRemover` method), 391  
`visit_for_iteration()`  
(`robot.result.keywordremover.ByTagKeywordRemover` method), 395  
`visit_for_iteration()`  
(`robot.result.keywordremover.ForLoopItemsRemover` method), 399  
`visit_for_iteration()`  
(`robot.result.keywordremover.PassedKeywordRemover` method), 387  
`visit_for_iteration()`  
(`robot.result.keywordremover.WaitUntilKeywordSucceedsRemover` method), 407  
`visit_for_iteration()`  
(`robot.result.keywordremover.WarningAndErrorFinder` method), 412  
`visit_for_iteration()`  
(`robot.result.keywordremover.WhileLoopItemsRemover` method), 403  
`visit_for_iteration()`  
(`robot.result.merger.Merger` method), 416  
`visit_for_iteration()`  
(`robot.result.messagefilter.MessageFilter` method), 420  
`visit_for_iteration()`  
(`robot.result.resultbuilder.RemoveKeywords` method), 468  
`visit_for_iteration()`  
(`robot.result.suitetardownfailed.SuiteTeardownFailed` method), 476  
`visit_for_iteration()`  
(`robot.result.suitetardownfailed.SuiteTeardownFailureHandler` method), 472  
`visit_for_iteration()`  
(`robot.result.visitor.ResultVisitor` method), 481  
`visit_for_iteration()`  
(`robot.running.randomizer.Randomizer` method), 544  
`visit_for_iteration()`  
(`robot.running.suiterunner.SuiteRunner` method), 550  
`visit_ForceTags()`  
(`robot.running.builder.transformers.SettingsBuilder` method), 506  
`visit_if()` (`robot.conf.gatherfailed.GatherFailedSuites` method), 30  
`visit_if()` (`robot.conf.gatherfailed.GatherFailedTests` method), 26  
`visit_if()` (`robot.model.configurer.SuiteConfigurer` method), 202



`visit_if()` (`robot.model.filter.EmptySuiteRemover` method), 218  
`visit_if()` (`robot.model.filter.Filter` method), 222  
`visit_if()` (`robot.model.modifier.ModelModifier` method), 234  
`visit_if()` (`robot.model.statistics.StatisticsBuilder` method), 238  
`visit_if()` (`robot.model.tagsetter.TagSetter` method), 246  
`visit_if()` (`robot.model.totalstatistics.TotalStatisticsBuilder` method), 257  
`visit_if()` (`robot.model.visitor.SuiteVisitor` method), 260  
`visit_if()` (`robot.output.console.dotted.StatusReporter` method), 267  
`visit_if()` (`robot.output.xmllogger.XmlLogger` method), 282  
`visit_if()` (`robot.reporting.outputwriter.OutputWriter` method), 363  
`visit_if()` (`robot.reporting.xunitwriter.XUnitFileWriter` method), 370  
`visit_if()` (`robot.result.configurer.SuiteConfigurer` method), 375  
`visit_if()` (`robot.result.keywordremover.AllKeywordsRemover` method), 383  
`visit_if()` (`robot.result.keywordremover.ByNameKeywordRemover` method), 391  
`visit_if()` (`robot.result.keywordremover.ByTagKeywordRemover` method), 395  
`visit_if()` (`robot.result.keywordremover.ForLoopItemsRemover` method), 399  
`visit_if()` (`robot.result.keywordremover.PassedKeywordRemover` method), 387  
`visit_if()` (`robot.result.keywordremover.WaitUntilKeywordSucceedsRemover` method), 408  
`visit_if()` (`robot.result.keywordremover.WarningAndErrorFinder` method), 412  
`visit_if()` (`robot.result.keywordremover.WhileLoopItemsRemover` method), 403  
`visit_if()` (`robot.result.merger.Merger` method), 416  
`visit_if()` (`robot.result.messagefilter.MessageFilter` method), 420  
`visit_if()` (`robot.result.resultbuilder.RemoveKeywords` method), 468  
`visit_if()` (`robot.result.suitetearndownfailed.SuiteTearndownFailed` method), 476  
`visit_if()` (`robot.result.suitetearndownfailed.SuiteTearndownFailureHandler` method), 472  
`visit_if()` (`robot.result.visitor.ResultVisitor` method), 481  
`visit_If()` (`robot.running.builder.transformers.ForBuilder` method), 508  
`visit_If()` (`robot.running.builder.transformers.IfBuilder` method), 509  
`visit_If()` (`robot.running.builder.transformers.KeywordBuilder` method), 508  
`visit_If()` (`robot.running.builder.transformers.TestCaseBuilder` method), 507  
`visit_If()` (`robot.running.builder.transformers.TryBuilder` method), 509  
`visit_If()` (`robot.running.builder.transformers.WhileBuilder` method), 510  
`visit_if()` (`robot.running.randomizer.Randomizer` method), 544  
`visit_if()` (`robot.running.suiterunner.SuiteRunner` method), 550  
`visit_if_branch()` (`robot.conf.gatherfailed.GatherFailedSuites` method), 30  
`visit_if_branch()` (`robot.conf.gatherfailed.GatherFailedTests` method), 26  
`visit_if_branch()` (`robot.model.configurer.SuiteConfigurer` method), 202  
`visit_if_branch()` (`robot.model.filter.EmptySuiteRemover` method), 218  
`visit_if_branch()` (`robot.model.filter.Filter` method), 223  
`visit_if_branch()` (`robot.model.modifier.ModelModifier` method), 234  
`visit_if_branch()` (`robot.model.statistics.StatisticsBuilder` method), 239  
`visit_if_branch()` (`robot.model.tagsetter.TagSetter` method), 246  
`visit_if_branch()` (`robot.model.totalstatistics.TotalStatisticsBuilder` method), 257  
`visit_if_branch()` (`robot.model.visitor.SuiteVisitor` method), 260  
`visit_if_branch()` (`robot.output.console.dotted.StatusReporter` method), 267  
`visit_if_branch()` (`robot.output.xmllogger.XmlLogger` method), 282  
`visit_if_branch()` (`robot.reporting.outputwriter.OutputWriter` method), 363  
`visit_if_branch()` (`robot.reporting.xunitwriter.XUnitFileWriter` method), 370

(*robot.result.configurer.SuiteConfigurer* method), 203  
 (*robot.result.configurer.SuiteConfigurer* method), 376  
 visit\_if\_branch() (*robot.result.keywordremover.AllKeywordsRemover* method), 379  
 visit\_if\_branch() (*robot.result.keywordremover.ByTagKeywordRemover* method), 395  
 visit\_if\_branch() (*robot.result.keywordremover.ByTagKeywordRemover* method), 395  
 visit\_if\_branch() (*robot.result.keywordremover.ForLoopItemsRemover* method), 399  
 visit\_if\_branch() (*robot.result.keywordremover.ForLoopItemsRemover* method), 399  
 visit\_if\_branch() (*robot.result.keywordremover.PassedKeywordRemover* method), 387  
 visit\_if\_branch() (*robot.result.keywordremover.PassedKeywordRemover* method), 387  
 visit\_if\_branch() (*robot.result.keywordremover.WaitUntilKeywordSucceedsRemover* method), 408  
 visit\_if\_branch() (*robot.result.keywordremover.WaitUntilKeywordSucceedsRemover* method), 408  
 visit\_if\_branch() (*robot.result.keywordremover.WarningAndErrorFinder* method), 412  
 visit\_if\_branch() (*robot.result.keywordremover.WarningAndErrorFinder* method), 412  
 visit\_if\_branch() (*robot.result.keywordremover.WhileLoopItemsRemover* method), 403  
 visit\_if\_branch() (*robot.result.keywordremover.WhileLoopItemsRemover* method), 403  
 visit\_if\_branch() (*robot.result.merger.Merger* method), 416  
 visit\_if\_branch() (*robot.result.merger.Merger* method), 416  
 visit\_if\_branch() (*robot.result.messagefilter.MessageFilter* method), 420  
 visit\_if\_branch() (*robot.result.messagefilter.MessageFilter* method), 420  
 visit\_if\_branch() (*robot.result.resultbuilder.RemoveKeywords* method), 468  
 visit\_if\_branch() (*robot.result.resultbuilder.RemoveKeywords* method), 468  
 visit\_if\_branch() (*robot.result.suiteteardownfailed.SuiteTeardownFailureHandler* method), 477  
 visit\_if\_branch() (*robot.result.suiteteardownfailed.SuiteTeardownFailureHandler* method), 477  
 visit\_if\_branch() (*robot.result.suiteteardownfailed.SuiteTeardownFailureHandler* method), 472  
 visit\_if\_branch() (*robot.result.suiteteardownfailed.SuiteTeardownFailureHandler* method), 472  
 visit\_if\_branch() (*robot.result.visitor.ResultVisitor* method), 482  
 visit\_if\_branch() (*robot.result.visitor.ResultVisitor* method), 482  
 visit\_if\_branch() (*robot.running.randomizer.Randomizer* method), 544  
 visit\_if\_branch() (*robot.running.randomizer.Randomizer* method), 544  
 visit\_if\_branch() (*robot.running.suiterunner.SuiteRunner* method), 550  
 visit\_if\_branch() (*robot.running.suiterunner.SuiteRunner* method), 550  
 visit\_keyword() (*robot.conf.gatherfailed.GatherFailedSuite* method), 27  
 visit\_keyword() (*robot.conf.gatherfailed.GatherFailedSuite* method), 27  
 visit\_keyword() (*robot.conf.gatherfailed.GatherFailedTest* method), 23  
 visit\_keyword() (*robot.conf.gatherfailed.GatherFailedTest* method), 23  
 visit\_keyword() (*robot.model.configurer.SuiteConfigurer* method), 203  
 visit\_keyword() (*robot.model.filter.EmptySuiteRemover* method), 215  
 visit\_keyword() (*robot.model.filter.Filter* method), 223  
 visit\_keyword() (*robot.model.filter.Filter* method), 223  
 visit\_keyword() (*robot.model.modifier.ModelModifier* method), 234  
 visit\_keyword() (*robot.model.modifier.ModelModifier* method), 234  
 visit\_keyword() (*robot.model.statistics.StatisticsBuilder* method), 235  
 visit\_keyword() (*robot.model.statistics.StatisticsBuilder* method), 235  
 visit\_keyword() (*robot.model.tagsetter.TagSetter* method), 242  
 visit\_keyword() (*robot.model.tagsetter.TagSetter* method), 242  
 visit\_keyword() (*robot.model.totalstatistics.TotalStatisticsBuilder* method), 253  
 visit\_keyword() (*robot.model.totalstatistics.TotalStatisticsBuilder* method), 253  
 visit\_keyword() (*robot.model.visitor.SuiteVisitor* method), 259  
 visit\_keyword() (*robot.model.visitor.SuiteVisitor* method), 259  
 visit\_keyword() (*robot.output.console.dotted.StatusReporter* method), 267  
 visit\_keyword() (*robot.output.console.dotted.StatusReporter* method), 267  
 visit\_keyword() (*robot.output.xmllogger.XmlLogger* method), 282  
 visit\_keyword() (*robot.output.xmllogger.XmlLogger* method), 282  
 visit\_keyword() (*robot.reporting.outputwriter.OutputWriter* method), 363  
 visit\_keyword() (*robot.reporting.outputwriter.OutputWriter* method), 363  
 visit\_keyword() (*robot.reporting.xunitwriter.XUnitFileWriter* method), 366  
 visit\_keyword() (*robot.reporting.xunitwriter.XUnitFileWriter* method), 366  
 visit\_keyword() (*robot.result.configurer.SuiteConfigurer* method), 376  
 visit\_keyword() (*robot.result.configurer.SuiteConfigurer* method), 376  
 visit\_keyword() (*robot.result.keywordremover.AllKeywordsRemover* method), 379  
 visit\_keyword() (*robot.result.keywordremover.AllKeywordsRemover* method), 379  
 visit\_keyword() (*robot.result.keywordremover.ByTagKeywordRemover* method), 395  
 visit\_keyword() (*robot.result.keywordremover.ByTagKeywordRemover* method), 395  
 visit\_keyword() (*robot.result.keywordremover.ByTagKeywordRemover* method), 395  
 visit\_keyword() (*robot.result.keywordremover.ByTagKeywordRemover* method), 395  
 visit\_keyword() (*robot.result.keywordremover.ForLoopItemsRemover* method), 399  
 visit\_keyword() (*robot.result.keywordremover.ForLoopItemsRemover* method), 399  
 visit\_keyword() (*robot.result.keywordremover.PassedKeywordRemover* method), 384  
 visit\_keyword() (*robot.result.keywordremover.PassedKeywordRemover* method), 384  
 visit\_keyword() (*robot.result.keywordremover.WaitUntilKeywordSucceedsRemover* method), 408  
 visit\_keyword() (*robot.result.keywordremover.WaitUntilKeywordSucceedsRemover* method), 408  
 visit\_keyword() (*robot.result.keywordremover.WaitUntilKeywordSucceedsRemover* method), 408  
 visit\_keyword() (*robot.result.keywordremover.WaitUntilKeywordSucceedsRemover* method), 408  
 visit\_keyword() (*robot.result.keywordremover.WarningAndErrorFinder* method), 412  
 visit\_keyword() (*robot.result.keywordremover.WarningAndErrorFinder* method), 412  
 visit\_keyword() (*robot.result.keywordremover.WarningAndErrorFinder* method), 412  
 visit\_keyword() (*robot.result.keywordremover.WarningAndErrorFinder* method), 412  
 visit\_keyword() (*robot.result.keywordremover.WhileLoopItemsRemover* method), 404  
 visit\_keyword() (*robot.result.keywordremover.WhileLoopItemsRemover* method), 404  
 visit\_keyword() (*robot.result.merger.Merger* method), 416  
 visit\_keyword() (*robot.result.merger.Merger* method), 416  
 visit\_keyword() (*robot.result.messagefilter.MessageFilter* method), 421  
 visit\_keyword() (*robot.result.messagefilter.MessageFilter* method), 421  
 visit\_keyword() (*robot.result.resultbuilder.RemoveKeywords* method), 468  
 visit\_keyword() (*robot.result.resultbuilder.RemoveKeywords* method), 468  
 visit\_keyword() (*robot.result.suiteteardownfailed.SuiteTeardownFailureHandler* method), 473  
 visit\_keyword() (*robot.result.suiteteardownfailed.SuiteTeardownFailureHandler* method), 473  
 visit\_keyword() (*robot.result.suiteteardownfailed.SuiteTeardownFailureHandler* method), 473  
 visit\_keyword() (*robot.result.suiteteardownfailed.SuiteTeardownFailureHandler* method), 473  
 visit\_keyword() (*robot.result.visitor.ResultVisitor* method), 482  
 visit\_keyword() (*robot.result.visitor.ResultVisitor* method), 482  
 visit\_keyword() (*robot.result.visitor.ResultVisitor* method), 482  
 visit\_keyword() (*robot.result.visitor.ResultVisitor* method), 482  
 visit\_keyword() (*robot.running.builder.transformers.KeywordBuilder* method), 550  
 visit\_keyword() (*robot.running.builder.transformers.KeywordBuilder* method), 550

*method*), 508  
 visit\_Keyword() (*robot.running.builder.transformers.ResourceBuilder*  
*method*), 507  
 visit\_Keyword() (*robot.running.builder.transformers.SuiteBuilder*  
*method*), 507  
 visit\_keyword() (*robot.running.randomizer.Randomizer*  
*method*), 541  
 visit\_keyword() (*robot.running.suiterunner.SuiteRunner*  
*method*), 550  
 visit\_KeywordCall()  
 (*robot.running.builder.transformers.ForBuilder*  
*method*), 508  
 visit\_KeywordCall()  
 (*robot.running.builder.transformers.IfBuilder*  
*method*), 509  
 visit\_KeywordCall()  
 (*robot.running.builder.transformers.KeywordBuilder*  
*method*), 508  
 visit\_KeywordCall()  
 (*robot.running.builder.transformers.TestCaseBuilder*  
*method*), 507  
 visit\_KeywordCall()  
 (*robot.running.builder.transformers.TryBuilder*  
*method*), 509  
 visit\_KeywordCall()  
 (*robot.running.builder.transformers.WhileBuilder*  
*method*), 509  
 visit\_KeywordSection()  
 (*robot.running.builder.transformers.SettingsBuilder*  
*method*), 506  
 visit\_LibraryImport()  
 (*robot.running.builder.transformers.ResourceBuilder*  
*method*), 507  
 visit\_LibraryImport()  
 (*robot.running.builder.transformers.SettingsBuilder*  
*method*), 506  
 visit\_message() (*robot.conf.gatherfailed.GatherFailedSuite*  
*method*), 30  
 visit\_message() (*robot.conf.gatherfailed.GatherFailedTest*  
*method*), 26  
 visit\_message() (*robot.model.configurer.SuiteConfigurer*  
*method*), 203  
 visit\_message() (*robot.model.filter.EmptySuiteRemover*  
*method*), 219  
 visit\_message() (*robot.model.filter.Filter* *method*),  
 223  
 visit\_message() (*robot.model.modifier.ModelModifier*  
*method*), 234  
 visit\_message() (*robot.model.statistics.StatisticsBuilder*  
*method*), 239  
 visit\_message() (*robot.model.tagsetter.TagSetter*  
*method*), 246  
 visit\_message() (*robot.model.totalstatistics.TotalStatisticsBuilder*  
*method*), 257  
 visit\_message() (*robot.model.visitor.SuiteVisitor*  
*method*), 263  
 visit\_message() (*robot.output.console.dotted.StatusReporter*  
*method*), 267  
 visit\_message() (*robot.output.xmllogger.XmlLogger*  
*method*), 282  
 visit\_message() (*robot.reporting.outputwriter.OutputWriter*  
*method*), 363  
 visit\_message() (*robot.reporting.xunitwriter.XUnitFileWriter*  
*method*), 370  
 visit\_message() (*robot.result.configurer.SuiteConfigurer*  
*method*), 376  
 visit\_message() (*robot.result.keywordremover.AllKeywordsRemover*  
*method*), 383  
 visit\_message() (*robot.result.keywordremover.ByNameKeywordRemover*  
*method*), 391  
 visit\_message() (*robot.result.keywordremover.ByTagKeywordRemover*  
*method*), 395  
 visit\_message() (*robot.result.keywordremover.ForLoopItemsRemover*  
*method*), 399  
 visit\_message() (*robot.result.keywordremover.PassedKeywordRemover*  
*method*), 387  
 visit\_message() (*robot.result.keywordremover.WaitUntilKeywordSuccess*  
*method*), 408  
 visit\_message() (*robot.result.keywordremover.WarningAndErrorFinder*  
*method*), 409  
 visit\_message() (*robot.result.keywordremover.WhileLoopItemsRemover*  
*method*), 404  
 visit\_message() (*robot.result.merger.Merger*  
*method*), 417  
 visit\_message() (*robot.result.messagefilter.MessageFilter*  
*method*), 421  
 visit\_message() (*robot.result.resultbuilder.RemoveKeywords*  
*method*), 468  
 visit\_message() (*robot.result.suiteteardownfailed.SuiteTeardownFailed*  
*method*), 477  
 visit\_message() (*robot.result.suiteteardownfailed.SuiteTeardownFailed*  
*method*), 473  
 visit\_message() (*robot.result.visitor.ResultVisitor*  
*method*), 482  
 visit\_message() (*robot.running.randomizer.Randomizer*  
*method*), 544  
 visit\_message() (*robot.running.suiterunner.SuiteRunner*  
*method*), 550  
 visit\_Metadata() (*robot.running.builder.transformers.SettingsBuilder*  
*method*), 506  
 visit\_ResourceImport()  
 (*robot.running.builder.transformers.ResourceBuilder*  
*method*), 507  
 visit\_ResourceImport()  
 (*robot.running.builder.transformers.SettingsBuilder*  
*method*), 506  
 visit\_result() (*robot.output.xmllogger.XmlLogger*  
*method*), 282

`visit_result()` (`robot.reporting.outputwriter.OutputWriter` method), 363

`visit_result()` (`robot.reporting.xunitwriter.XUnitFileWriter` method), 370

`visit_result()` (`robot.result.visitor.ResultVisitor` method), 478

`visit_return()` (`robot.conf.gatherfailed.GatherFailedSuite` method), 31

`visit_return()` (`robot.conf.gatherfailed.GatherFailedTests` method), 26

`visit_return()` (`robot.model.configurer.SuiteConfigurer` method), 203

`visit_return()` (`robot.model.filter.EmptySuiteRemover` method), 219

`visit_return()` (`robot.model.filter.Filter` method), 223

`visit_return()` (`robot.model.modifier.ModelModifier` method), 234

`visit_return()` (`robot.model.statistics.StatisticsBuilder` method), 239

`visit_return()` (`robot.model.tagsetter.TagSetter` method), 246

`visit_return()` (`robot.model.totalstatistics.TotalStatisticsBuilder` method), 257

`visit_return()` (`robot.model.visitor.SuiteVisitor` method), 262

`visit_return()` (`robot.output.console.dotted.StatusReporter` method), 267

`visit_return()` (`robot.output.xmllogger.XmlLogger` method), 282

`visit_return()` (`robot.reporting.outputwriter.OutputWriter` method), 363

`visit_return()` (`robot.reporting.xunitwriter.XUnitFileWriter` method), 370

`visit_return()` (`robot.result.configurer.SuiteConfigurer` method), 376

`visit_return()` (`robot.result.keywordremover.AllKeywordsRemover` method), 383

`visit_return()` (`robot.result.keywordremover.ByTagNameKeywordRemover` method), 391

`visit_return()` (`robot.result.keywordremover.ByTagKeywordRemover` method), 395

`visit_return()` (`robot.result.keywordremover.ForLoopItemsRemover` method), 400

`visit_return()` (`robot.result.keywordremover.PassedKeywordRemover` method), 387

`visit_return()` (`robot.result.keywordremover.WaitUntilKeywordSuccesfullyRemover` method), 408

`visit_return()` (`robot.result.keywordremover.WarningAndErrorFinder` method), 412

`visit_return()` (`robot.result.keywordremover.WhileLoopItemsRemover` method), 404

`visit_return()` (`robot.result.merger.Merger` method), 417

`visit_return()` (`robot.result.messagefilter.MessageFilter` method), 421

`visit_return()` (`robot.result.resultbuilder.RemoveKeywords` method), 468

`visit_return()` (`robot.result.suite teardownfailed.SuiteTeardownFailed` method), 477

`visit_return()` (`robot.result.suite teardownfailed.SuiteTeardownFailure` method), 473

`visit_return()` (`robot.result.visitor.ResultVisitor` method), 482

`visit_return()` (`robot.running.builder.transformers.KeywordBuilder` method), 508

`visit_return()` (`robot.running.randomizer.Randomizer` method), 544

`visit_return()` (`robot.running.suiterunner.SuiteRunner` method), 550

`visit_ReturnStatement()` (`robot.running.builder.transformers.ForBuilder` method), 508

`visit_ReturnStatement()` (`robot.running.builder.transformers.IfBuilder` method), 509

`visit_ReturnStatement()` (`robot.running.builder.transformers.KeywordBuilder` method), 508

`visit_ReturnStatement()` (`robot.running.builder.transformers.TestCaseBuilder` method), 507

`visit_ReturnStatement()` (`robot.running.builder.transformers.TryBuilder` method), 509

`visit_ReturnStatement()` (`robot.running.builder.transformers.WhileBuilder` method), 510

`visit_SettingSection()` (`robot.running.builder.transformers.SuiteBuilder` method), 506

`visit_Setup()` (`robot.running.builder.transformers.TestCaseBuilder` method), 507

`visit_stat()` (`robot.output.xmllogger.XmlLogger` method), 281

`visit_stat()` (`robot.reporting.outputwriter.OutputWriter` method), 364

`visit_stat()` (`robot.reporting.xunitwriter.XUnitFileWriter` method), 370

`visit_stat()` (`robot.result.visitor.ResultVisitor` method), 482

`visit_Statement()` (`robot.parsing.model.blocks.FirstStatementFinder` method), 311

`visit_Statement()` (`robot.parsing.model.blocks.LastStatementFinder` method), 311



(*robot.parsing.model.blocks.ModelValidator*  
*method*), 310  
 visit\_Statement() (*robot.parsing.model.blocks.ModelWriter*  
*method*), 310  
 visit\_statistics() (*robot.output.xmllogger.XmlLogger* *method*),  
 282  
 visit\_statistics() (*robot.reporting.outputwriter.OutputWriter*  
*method*), 364  
 visit\_statistics() (*robot.reporting.xunitwriter.XUnitFileWriter*  
*method*), 366  
 visit\_statistics() (*robot.result.visitor.ResultVisitor* *method*),  
 478  
 visit\_suite() (*robot.conf.gatherfailed.GatherFailedSuites*  
*method*), 31  
 visit\_suite() (*robot.conf.gatherfailed.GatherFailedTests*  
*method*), 26  
 visit\_suite() (*robot.model.configurer.SuiteConfigurer*  
*method*), 199  
 visit\_suite() (*robot.model.filter.EmptySuiteRemover*  
*method*), 219  
 visit\_suite() (*robot.model.filter.Filter* *method*),  
 223  
 visit\_suite() (*robot.model.modifier.ModelModifier*  
*method*), 230  
 visit\_suite() (*robot.model.statistics.StatisticsBuilder*  
*method*), 239  
 visit\_suite() (*robot.model.tagsetter.TagSetter*  
*method*), 246  
 visit\_suite() (*robot.model.totalstatistics.TotalStatisticsBuilder*  
*method*), 257  
 visit\_suite() (*robot.model.visitor.SuiteVisitor*  
*method*), 259  
 visit\_suite() (*robot.output.console.dotted.StatusReporter*  
*method*), 267  
 visit\_suite() (*robot.output.xmllogger.XmlLogger*  
*method*), 282  
 visit\_suite() (*robot.reporting.outputwriter.OutputWriter*  
*method*), 364  
 visit\_suite() (*robot.reporting.xunitwriter.XUnitFileWriter*  
*method*), 370  
 visit\_suite() (*robot.result.configurer.SuiteConfigurer*  
*method*), 372  
 visit\_suite() (*robot.result.keywordremover.AllKeywordsRemover*  
*method*), 383  
 visit\_suite() (*robot.result.keywordremover.ByNameKeywordRemover*  
*method*), 391  
 visit\_suite() (*robot.result.keywordremover.ByTagKeywordRemover*  
*method*), 395  
 visit\_suite() (*robot.result.keywordremover.ForLoopItemsRemover*  
*method*), 400  
 visit\_suite() (*robot.result.keywordremover.PassedKeywordRemover*  
*method*), 387  
 visit\_suite() (*robot.result.keywordremover.WaitUntilKeywordSucceeds*  
*method*), 408  
 visit\_suite() (*robot.result.keywordremover.WarningAndErrorFinder*  
*method*), 412  
 visit\_suite() (*robot.result.keywordremover.WhileLoopItemsRemover*  
*method*), 404  
 visit\_suite() (*robot.result.merger.Merger* *method*),  
 417  
 visit\_suite() (*robot.result.messagefilter.MessageFilter*  
*method*), 421  
 visit\_suite() (*robot.result.resultbuilder.RemoveKeywords*  
*method*), 468  
 visit\_suite() (*robot.result.suite teardownfailed.SuiteTeardownFailed*  
*method*), 477  
 visit\_suite() (*robot.result.suite teardownfailed.SuiteTeardownFailure*  
*method*), 473  
 visit\_suite() (*robot.result.visitor.ResultVisitor*  
*method*), 482  
 visit\_suite() (*robot.running.randomizer.Randomizer*  
*method*), 544  
 visit\_suite() (*robot.running.suiterunner.SuiteRunner*  
*method*), 550  
 visit\_suite\_statistics() (*robot.output.xmllogger.XmlLogger* *method*),  
 282  
 visit\_suite\_statistics() (*robot.reporting.outputwriter.OutputWriter*  
*method*), 364  
 visit\_suite\_statistics() (*robot.reporting.xunitwriter.XUnitFileWriter*  
*method*), 370  
 visit\_suite\_statistics() (*robot.result.visitor.ResultVisitor* *method*),  
 478  
 visit\_SuiteSetup() (*robot.running.builder.transformers.SettingsBuilder*  
*method*), 506  
 visit\_SuiteTeardown() (*robot.running.builder.transformers.SettingsBuilder*  
*method*), 506  
 visit\_tag\_statistics() (*robot.output.xmllogger.XmlLogger* *method*),  
 282  
 visit\_tag\_statistics() (*robot.reporting.outputwriter.OutputWriter*  
*method*), 364  
 visit\_tag\_statistics() (*robot.reporting.xunitwriter.XUnitFileWriter*  
*method*), 370  
 visit\_tag\_statistics() (*robot.result.visitor.ResultVisitor* *method*),  
 478

478

`visit_Tags()` (`robot.running.builder.transformers.KeywordBuilder` method), 508

`visit_Tags()` (`robot.running.builder.transformers.TestCaseBuilder` method), 507

`visit_Teardown()` (`robot.running.builder.transformers.KeywordBuilder` method), 508

`visit_Teardown()` (`robot.running.builder.transformers.TestCaseBuilder` method), 507

`visit_Template()` (`robot.running.builder.transformers.TestCaseBuilder` method), 507

`visit_TemplateArguments()` (`robot.running.builder.transformers.ForBuilder` method), 508

`visit_TemplateArguments()` (`robot.running.builder.transformers.IfBuilder` method), 509

`visit_TemplateArguments()` (`robot.running.builder.transformers.TestCaseBuilder` method), 507

`visit_TemplateArguments()` (`robot.running.builder.transformers.TryBuilder` method), 509

`visit_TemplateArguments()` (`robot.running.builder.transformers.WhileBuilder` method), 510

`visit_test()` (`robot.conf.gatherfailed.GatherFailedSuites` method), 27

`visit_test()` (`robot.conf.gatherfailed.GatherFailedTests` method), 23

`visit_test()` (`robot.model.configurer.SuiteConfigurer` method), 203

`visit_test()` (`robot.model.filter.EmptySuiteRemover` method), 215

`visit_test()` (`robot.model.filter.Filter` method), 223

`visit_test()` (`robot.model.modifier.ModelModifier` method), 234

`visit_test()` (`robot.model.statistics.StatisticsBuilder` method), 235

`visit_test()` (`robot.model.tagsetter.TagSetter` method), 242

`visit_test()` (`robot.model.totalstatistics.TotalStatisticsBuilder` method), 253

`visit_test()` (`robot.model.visitor.SuiteVisitor` method), 259

`visit_test()` (`robot.output.console.dotted.StatusReporter` method), 264

`visit_test()` (`robot.output.xmllogger.XmlLogger` method), 282

`visit_test()` (`robot.reporting.outputwriter.OutputWriter` method), 364

`visit_test()` (`robot.reporting.xunitwriter.XUnitFileWriter` method), 366

`visit_test()` (`robot.result.configurer.SuiteConfigurer` method), 376

`visit_test()` (`robot.result.keywordremover.AllKeywordsRemover` method), 383

`visit_test()` (`robot.result.keywordremover.ByTagKeywordRemover` method), 391

`visit_test()` (`robot.result.keywordremover.ByTagKeywordRemover` method), 395

`visit_test()` (`robot.result.keywordremover.ForLoopItemsRemover` method), 400

`visit_test()` (`robot.result.keywordremover.PassedKeywordRemover` method), 384

`visit_test()` (`robot.result.keywordremover.WaitUntilKeywordSucceeds` method), 408

`visit_test()` (`robot.result.keywordremover.WarningAndErrorFinder` method), 412

`visit_test()` (`robot.result.keywordremover.WhileLoopItemsRemover` method), 404

`visit_test()` (`robot.result.merger.Merger` method), 413

`visit_test()` (`robot.result.messagefilter.MessageFilter` method), 421

`visit_test()` (`robot.result.resultbuilder.RemoveKeywords` method), 465

`visit_test()` (`robot.result.suiteteardownfailed.SuiteTeardownFailed` method), 473

`visit_test()` (`robot.result.suiteteardownfailed.SuiteTeardownFailureH` method), 469

`visit_test()` (`robot.result.visitor.ResultVisitor` method), 482

`visit_test()` (`robot.running.randomizer.Randomizer` method), 541

`visit_test()` (`robot.running.suiterunner.SuiteRunner` method), 547

`visit_TestCase()` (`robot.running.builder.transformers.SuiteBuilder` method), 507

`visit_TestCase()` (`robot.running.builder.transformers.TestCaseBuilder` method), 507

`visit_TestCaseSection()` (`robot.running.builder.transformers.SettingsBuilder` method), 506

`visit_TestSetup()` (`robot.running.builder.transformers.SettingsBuilder` method), 506

`visit_TestTeardown()` (`robot.running.builder.transformers.SettingsBuilder` method), 506

`visit_TestTemplate()` (`robot.running.builder.transformers.SettingsBuilder` method), 506

`visit_TestTimeout()` (`robot.running.builder.transformers.SettingsBuilder` method), 506

`visit_Timeout()` (`robot.running.builder.transformers.KeywordBuilder` method), 508

`visit_timeout()` (`robot.running.builder.transformers.TestCaseBuilder` method), 387  
`visit_total_statistics()` (`robot.output.xmllogger.XmlLogger` method), 283  
`visit_total_statistics()` (`robot.reporting.outputwriter.OutputWriter` method), 364  
`visit_total_statistics()` (`robot.reporting.xunitwriter.XUnitFileWriter` method), 370  
`visit_total_statistics()` (`robot.result.visitor.ResultVisitor` method), 478  
`visit_try()` (`robot.conf.gatherfailed.GatherFailedSuites` method), 31  
`visit_try()` (`robot.conf.gatherfailed.GatherFailedTests` method), 26  
`visit_try()` (`robot.model.configurer.SuiteConfigurer` method), 203  
`visit_try()` (`robot.model.filter.EmptySuiteRemover` method), 219  
`visit_try()` (`robot.model.filter.Filter` method), 223  
`visit_try()` (`robot.model.modifier.ModelModifier` method), 234  
`visit_try()` (`robot.model.statistics.StatisticsBuilder` method), 239  
`visit_try()` (`robot.model.tagsetter.TagSetter` method), 246  
`visit_try()` (`robot.model.totalstatistics.TotalStatisticsBuilder` method), 257  
`visit_try()` (`robot.model.visitor.SuiteVisitor` method), 261  
`visit_try()` (`robot.output.console.dotted.StatusReporter` method), 267  
`visit_try()` (`robot.output.xmllogger.XmlLogger` method), 283  
`visit_Try()` (`robot.parsing.model.blocks.ModelValidator` method), 310  
`visit_try()` (`robot.reporting.outputwriter.OutputWriter` method), 364  
`visit_try()` (`robot.reporting.xunitwriter.XUnitFileWriter` method), 370  
`visit_try()` (`robot.result.configurer.SuiteConfigurer` method), 376  
`visit_try()` (`robot.result.keywordremover.AllKeywordsRemover` method), 383  
`visit_try()` (`robot.result.keywordremover.ByNameKeywordRemover` method), 391  
`visit_try()` (`robot.result.keywordremover.ByTagKeywordRemover` method), 396  
`visit_try()` (`robot.result.keywordremover.ForLoopItemsRemover` method), 400  
`visit_try()` (`robot.result.keywordremover.PassedKeywordRemover` method), 408  
`visit_try()` (`robot.result.keywordremover.WaitUntilKeywordSucceedsRemover` method), 412  
`visit_try()` (`robot.result.keywordremover.WarningAndErrorFinder` method), 412  
`visit_try()` (`robot.result.keywordremover.WhileLoopItemsRemover` method), 404  
`visit_try()` (`robot.result.merger.Merger` method), 417  
`visit_try()` (`robot.result.messagefilter.MessageFilter` method), 421  
`visit_try()` (`robot.result.resultbuilder.RemoveKeywords` method), 468  
`visit_try()` (`robot.result.suite teardownfailed.SuiteTeardownFailed` method), 477  
`visit_try()` (`robot.result.suite teardownfailed.SuiteTeardownFailureHandler` method), 473  
`visit_try()` (`robot.result.visitor.ResultVisitor` method), 482  
`visit_Try()` (`robot.running.builder.transformers.ForBuilder` method), 508  
`visit_Try()` (`robot.running.builder.transformers.IfBuilder` method), 509  
`visit_Try()` (`robot.running.builder.transformers.KeywordBuilder` method), 508  
`visit_Try()` (`robot.running.builder.transformers.TestCaseBuilder` method), 507  
`visit_Try()` (`robot.running.builder.transformers.TryBuilder` method), 509  
`visit_Try()` (`robot.running.builder.transformers.WhileBuilder` method), 510  
`visit_try()` (`robot.running.randomizer.Randomizer` method), 544  
`visit_try()` (`robot.running.suiterunner.SuiteRunner` method), 551  
`visit_try_branch()` (`robot.conf.gatherfailed.GatherFailedSuites` method), 31  
`visit_try_branch()` (`robot.conf.gatherfailed.GatherFailedTests` method), 27  
`visit_try_branch()` (`robot.model.configurer.SuiteConfigurer` method), 203  
`visit_try_branch()` (`robot.model.filter.EmptySuiteRemover` method), 219  
`visit_try_branch()` (`robot.model.filter.Filter` method), 223  
`visit_try_branch()` (`robot.model.modifier.ModelModifier` method), 234  
`visit_try_branch()` (`robot.model.statistics.StatisticsBuilder` method), 239

*method*), 239

`visit_try_branch()`  
(*robot.model.tagsetter.TagSetter* *method*), 246

`visit_try_branch()`  
(*robot.model.totalstatistics.TotalStatisticsBuilder* *method*), 257

`visit_try_branch()`  
(*robot.model.visitor.SuiteVisitor* *method*), 261

`visit_try_branch()`  
(*robot.output.console.dotted.StatusReporter* *method*), 268

`visit_try_branch()`  
(*robot.output.xmllogger.XmlLogger* *method*), 283

`visit_try_branch()`  
(*robot.reporting.outputwriter.OutputWriter* *method*), 364

`visit_try_branch()`  
(*robot.reporting.xunitwriter.XUnitFileWriter* *method*), 371

`visit_try_branch()`  
(*robot.result.configurer.SuiteConfigurer* *method*), 376

`visit_try_branch()`  
(*robot.result.keywordremover.AllKeywordsRemover* *method*), 383

`visit_try_branch()`  
(*robot.result.keywordremover.ByNameKeywordRemover* *method*), 391

`visit_try_branch()`  
(*robot.result.keywordremover.ByTagKeywordRemover* *method*), 396

`visit_try_branch()`  
(*robot.result.keywordremover.ForLoopItemsRemover* *method*), 400

`visit_try_branch()`  
(*robot.result.keywordremover.PassedKeywordRemover* *method*), 387

`visit_try_branch()`  
(*robot.result.keywordremover.WaitUntilKeywordSucceedsRemover* *method*), 408

`visit_try_branch()`  
(*robot.result.keywordremover.WarningAndErrorFinder* *method*), 412

`visit_try_branch()`  
(*robot.result.keywordremover.WhileLoopItemsRemover* *method*), 404

`visit_try_branch()` (*robot.result.merger.Merger* *method*), 417

`visit_try_branch()`  
(*robot.result.messagefilter.MessageFilter* *method*), 421

`visit_try_branch()`  
(*robot.result.resultbuilder.RemoveKeywords* *method*), 469

`visit_try_branch()`  
(*robot.result.suite teardownfailed.SuiteTeardownFailed* *method*), 477

`visit_try_branch()`  
(*robot.result.suite teardownfailed.SuiteTeardownFailureHandler* *method*), 473

`visit_try_branch()`  
(*robot.result.visitor.ResultVisitor* *method*), 482

`visit_try_branch()`  
(*robot.running.randomizer.Randomizer* *method*), 544

`visit_try_branch()`  
(*robot.running.suite runner.SuiteRunner* *method*), 551

`visit_Variable()` (*robot.running.builder.transformers.ResourceBuilder* *method*), 507

`visit_Variable()` (*robot.running.builder.transformers.SuiteBuilder* *method*), 506

`visit_VariableSection()`  
(*robot.running.builder.transformers.SettingsBuilder* *method*), 506

`visit_VariablesImport()`  
(*robot.running.builder.transformers.ResourceBuilder* *method*), 507

`visit_VariablesImport()`  
(*robot.running.builder.transformers.SettingsBuilder* *method*), 506

`visit_while()` (*robot.conf.gatherfailed.GatherFailedSuites* *method*), 31

`visit_while()` (*robot.conf.gatherfailed.GatherFailedTests* *method*), 27

`visit_while()` (*robot.model.configurer.SuiteConfigurer* *method*), 203

`visit_while()` (*robot.model.filter.EmptySuiteRemover* *method*), 219

`visit_while()` (*robot.model.filter.Filter* *method*), 223

`visit_while()` (*robot.model.modifier.ModelModifier* *method*), 234

`visit_while()` (*robot.model.statistics.StatisticsBuilder* *method*), 239

`visit_while()` (*robot.model.tagsetter.TagSetter* *method*), 246

`visit_while()` (*robot.model.totalstatistics.TotalStatisticsBuilder* *method*), 257

`visit_while()` (*robot.model.visitor.SuiteVisitor* *method*), 261

`visit_while()` (*robot.output.console.dotted.StatusReporter* *method*), 268

`visit_while()` (*robot.output.xmllogger.XmlLogger*

[method](#)), 283  
[visit\\_while\(\)](#) ([robot.reporting.outputwriter.OutputWriter](#) [method](#)), 364  
[visit\\_while\(\)](#) ([robot.reporting.xunitwriter.XUnitFileWriter](#) [method](#)), 371  
[visit\\_while\(\)](#) ([robot.result.configurer.SuiteConfigurer](#) [method](#)), 376  
[visit\\_while\(\)](#) ([robot.result.keywordremover.AllKeywordsRemover](#) [method](#)), 383  
[visit\\_while\(\)](#) ([robot.result.keywordremover.ByTagNameKeywordRemover](#) [method](#)), 392  
[visit\\_while\(\)](#) ([robot.result.keywordremover.ByTagKeywordRemover](#) [method](#)), 396  
[visit\\_while\(\)](#) ([robot.result.keywordremover.ForLoopItemsRemover](#) [method](#)), 400  
[visit\\_while\(\)](#) ([robot.result.keywordremover.PassedKeywordRemover](#) [method](#)), 387  
[visit\\_while\(\)](#) ([robot.result.keywordremover.WaitUntilKeywordSucceedsRemover](#) [method](#)), 408  
[visit\\_while\(\)](#) ([robot.result.keywordremover.WarningAndErrorFinder](#) [method](#)), 413  
[visit\\_while\(\)](#) ([robot.result.keywordremover.WhileLoopItemsRemover](#) [method](#)), 404  
[visit\\_while\(\)](#) ([robot.result.merger.Merger](#) [method](#)), 417  
[visit\\_while\(\)](#) ([robot.result.messagefilter.MessageFilter](#) [method](#)), 421  
[visit\\_while\(\)](#) ([robot.result.resultbuilder.RemoveKeywords](#) [method](#)), 469  
[visit\\_while\(\)](#) ([robot.result.suiteteardownfailed.SuiteTeardownFailed](#) [method](#)), 477  
[visit\\_while\(\)](#) ([robot.result.suiteteardownfailed.SuiteTeardownFailedHandler](#) [method](#)), 473  
[visit\\_while\(\)](#) ([robot.result.visitor.ResultVisitor](#) [method](#)), 482  
[visit\\_While\(\)](#) ([robot.running.builder.transformers.ForBuilder](#) [method](#)), 508  
[visit\\_While\(\)](#) ([robot.running.builder.transformers.IfBuilder](#) [method](#)), 509  
[visit\\_While\(\)](#) ([robot.running.builder.transformers.KeywordBuilder](#) [method](#)), 508  
[visit\\_While\(\)](#) ([robot.running.builder.transformers.TestCaseBuilder](#) [method](#)), 507  
[visit\\_While\(\)](#) ([robot.running.builder.transformers.TryBuilder](#) [method](#)), 509  
[visit\\_While\(\)](#) ([robot.running.builder.transformers.WhileBuilder](#) [method](#)), 510  
[visit\\_while\(\)](#) ([robot.running.randomizer.Randomizer](#) [method](#)), 544  
[visit\\_while\(\)](#) ([robot.running.suiterunner.SuiteRunner](#) [method](#)), 551  
[visit\\_while\\_iteration\(\)](#) ([robot.conf.gatherfailed.GatherFailedSuites](#) [method](#)), 31  
[visit\\_while\\_iteration\(\)](#) ([robot.conf.gatherfailed.GatherFailedTests](#) [method](#)), 27  
[visit\\_while\\_iteration\(\)](#) ([robot.model.configurer.SuiteConfigurer](#) [method](#)), 203  
[visit\\_while\\_iteration\(\)](#) ([robot.model.filter.EmptySuiteRemover](#) [method](#)), 219  
[visit\\_while\\_iteration\(\)](#) ([robot.model.filter.Filter](#) [method](#)), 223  
[visit\\_while\\_iteration\(\)](#) ([robot.model.modifier.ModelModifier](#) [method](#)), 224  
[visit\\_while\\_iteration\(\)](#) ([robot.model.statistics.StatisticsBuilder](#) [method](#)), 239  
[visit\\_while\\_iteration\(\)](#) ([robot.model.tagsetter.TagSetter](#) [method](#)), 246  
[visit\\_while\\_iteration\(\)](#) ([robot.model.totalstatistics.TotalStatisticsBuilder](#) [method](#)), 257  
[visit\\_while\\_iteration\(\)](#) ([robot.model.visitor.SuiteVisitor](#) [method](#)), 262  
[visit\\_while\\_iteration\(\)](#) ([robot.output.console.dotted.StatusReporter](#) [method](#)), 268  
[visit\\_while\\_iteration\(\)](#) ([robot.output.xmllogger.XmlLogger](#) [method](#)), 283



(*robot.result.keywordremover.PassedKeywordRemover* method), 387  
 visit\_while\_iteration() (*robot.result.keywordremover.WaitUntilKeywordSucceedsRemover* method), 408  
 visit\_while\_iteration() (*robot.result.keywordremover.WarningAndErrorFinder* method), 413  
 visit\_while\_iteration() (*robot.result.keywordremover.WhileLoopItemsRemover* method), 404  
 visit\_while\_iteration() (*robot.result.merger.Merger* method), 417  
 visit\_while\_iteration() (*robot.result.messagefilter.MessageFilter* method), 421  
 visit\_while\_iteration() (*robot.result.resultbuilder.RemoveKeywords* method), 469  
 visit\_while\_iteration() (*robot.result.suitetardownfailed.SuiteTeardownFailed* method), 477  
 visit\_while\_iteration() (*robot.result.suitetardownfailed.SuiteTeardownFailedHandler* method), 473  
 visit\_while\_iteration() (*robot.result.visitor.ResultVisitor* method), 482  
 visit\_while\_iteration() (*robot.running.randomizer.Randomizer* method), 545  
 visit\_while\_iteration() (*robot.running.suiterunner.SuiteRunner* method), 551  
 VisitorFinder (class *robot.parsing.model.visitor*), 353  
**W**  
 wait\_for\_process() (*robot.libraries.Process.Process* method), 91  
 wait\_until\_created() (*robot.libraries.OperatingSystem.OperatingSystem* method), 81  
 wait\_until\_keyword\_succeeds() (*robot.libraries.BuiltIn.BuiltIn* method), 64  
 wait\_until\_removed() (*robot.libraries.OperatingSystem.OperatingSystem* method), 81  
 wait\_variable() (*robot.libraries.dialogs\_py.InputDialog* method), 145  
 wait\_variable() (*robot.libraries.dialogs\_py.MessageDialog* method), 131  
 wait\_variable() (*robot.libraries.dialogs\_py.MultipleSelectionDialog* method), 173  
 wait\_variable() (*robot.libraries.dialogs\_py.PassFailDialog* method), 187  
 wait\_variable() (*robot.libraries.dialogs\_py.SelectionDialog* method), 159  
 wait\_visibility() (*robot.libraries.dialogs\_py.InputDialog* method), 145  
 wait\_visibility() (*robot.libraries.dialogs\_py.MessageDialog* method), 131  
 wait\_visibility() (*robot.libraries.dialogs\_py.MultipleSelectionDialog* method), 173  
 wait\_visibility() (*robot.libraries.dialogs\_py.PassFailDialog* method), 187  
 wait\_visibility() (*robot.libraries.dialogs\_py.SelectionDialog* method), 159  
 wait\_window() (*robot.libraries.dialogs\_py.InputDialog* method), 145  
 wait\_window() (*robot.libraries.dialogs\_py.MessageDialog* method), 132  
 wait\_window() (*robot.libraries.dialogs\_py.MultipleSelectionDialog* method), 173  
 wait\_window() (*robot.libraries.dialogs\_py.PassFailDialog* method), 187  
 wait\_window() (*robot.libraries.dialogs\_py.SelectionDialog* method), 159  
 waiting\_item\_state() (*robot.variables.search.VariableSearcher* method), 573  
 in WaitUntilKeywordSucceedsRemover (class in *robot.result.keywordremover*), 404  
 waitvar() (*robot.libraries.dialogs\_py.InputDialog* method), 146  
 waitvar() (*robot.libraries.dialogs\_py.MessageDialog* method), 132  
 waitvar() (*robot.libraries.dialogs\_py.MultipleSelectionDialog* method), 174  
 waitvar() (*robot.libraries.dialogs\_py.PassFailDialog* method), 188  
 waitvar() (*robot.libraries.dialogs\_py.SelectionDialog* method), 160  
 warn() (in module *robot.api.logger*), 15  
 warn() (in module *robot.output.librarylogger*), 270  
 warn() (*robot.output.filelogger.FileLogger* method), 270  
 warn() (*robot.output.logger.Logger* method), 274  
 warn() (*robot.output.loggerhelper.AbstractLogger* method), 274  
 warn() (*robot.output.output.Output* method), 276

- `warn()` (*robot.utils.importer.NoLogger* method), 564  
`WarningAndErrorFinder` (class in *robot.result.keywordremover*), 409  
`While` (class in *robot.model.control*), 205  
`While` (class in *robot.parsing.model.blocks*), 310  
`While` (class in *robot.result.model*), 435  
`While` (class in *robot.running.model*), 521  
`WHILE` (*robot.model.body.BodyItem* attribute), 193  
`WHILE` (*robot.model.control.Break* attribute), 214  
`WHILE` (*robot.model.control.Continue* attribute), 213  
`WHILE` (*robot.model.control.For* attribute), 204  
`WHILE` (*robot.model.control.If* attribute), 208  
`WHILE` (*robot.model.control.IfBranch* attribute), 207  
`WHILE` (*robot.model.control.Return* attribute), 212  
`WHILE` (*robot.model.control.Try* attribute), 211  
`WHILE` (*robot.model.control.TryBranch* attribute), 209  
`WHILE` (*robot.model.control.While* attribute), 205  
`WHILE` (*robot.model.keyword.Keyword* attribute), 226  
`WHILE` (*robot.model.message.Message* attribute), 228  
`WHILE` (*robot.output.loggerhelper.Message* attribute), 275  
`WHILE` (*robot.parsing.lexer.tokens.END* attribute), 305  
`WHILE` (*robot.parsing.lexer.tokens.EOS* attribute), 303  
`WHILE` (*robot.parsing.lexer.tokens.Token* attribute), 300  
`WHILE` (*robot.result.model.Break* attribute), 452  
`WHILE` (*robot.result.model.Continue* attribute), 450  
`WHILE` (*robot.result.model.For* attribute), 432  
`WHILE` (*robot.result.model.ForIteration* attribute), 429  
`WHILE` (*robot.result.model.If* attribute), 441  
`WHILE` (*robot.result.model.IfBranch* attribute), 439  
`WHILE` (*robot.result.model.Keyword* attribute), 455  
`WHILE` (*robot.result.model.Message* attribute), 427  
`WHILE` (*robot.result.model.Return* attribute), 448  
`WHILE` (*robot.result.model.Try* attribute), 445  
`WHILE` (*robot.result.model.TryBranch* attribute), 443  
`WHILE` (*robot.result.model.While* attribute), 436  
`WHILE` (*robot.result.model.WhileIteration* attribute), 434  
`WHILE` (*robot.running.model.Break* attribute), 531  
`WHILE` (*robot.running.model.Continue* attribute), 530  
`WHILE` (*robot.running.model.For* attribute), 520  
`WHILE` (*robot.running.model.If* attribute), 525  
`WHILE` (*robot.running.model.IfBranch* attribute), 523  
`WHILE` (*robot.running.model.Keyword* attribute), 518  
`WHILE` (*robot.running.model.Return* attribute), 529  
`WHILE` (*robot.running.model.Try* attribute), 527  
`WHILE` (*robot.running.model.TryBranch* attribute), 526  
`WHILE` (*robot.running.model.While* attribute), 522  
`while_class` (*robot.model.body.BaseBody* attribute), 194  
`while_class` (*robot.model.body.Body* attribute), 197  
`while_class` (*robot.model.body.Branches* attribute), 199  
`while_class` (*robot.result.model.Body* attribute), 423  
`while_class` (*robot.result.model.Branches* attribute), 425  
`while_class` (*robot.result.model.Iterations* attribute), 427  
`while_class` (*robot.running.model.Body* attribute), 517  
`WhileBuilder` (class in *robot.running.builder.transformers*), 509  
`WhileHandler` (class in *robot.result.xmllelementhandlers*), 484  
`WhileHeader` (class in *robot.parsing.model.statements*), 346  
`WhileHeaderLexer` (class in *robot.parsing.lexer.statementlexers*), 298  
`WhileIteration` (class in *robot.result.model*), 433  
`WhileLexer` (class in *robot.parsing.lexer.blocklexers*), 287  
`WhileLimit` (class in *robot.running.bodyrunner*), 512  
`WhileLoopItemsRemover` (class in *robot.result.keywordremover*), 400  
`WhileParser` (class in *robot.parsing.parser.blockparsers*), 354  
`WhileRunner` (class in *robot.running.bodyrunner*), 512  
`winfo_atom()` (*robot.libraries.dialogs\_py.InputDialog* method), 146  
`winfo_atom()` (*robot.libraries.dialogs\_py.MessageDialog* method), 132  
`winfo_atom()` (*robot.libraries.dialogs\_py MultipleSelectionDialog* method), 174  
`winfo_atom()` (*robot.libraries.dialogs\_py.PassFailDialog* method), 188  
`winfo_atom()` (*robot.libraries.dialogs\_py.SelectionDialog* method), 160  
`winfo_atomname()` (*robot.libraries.dialogs\_py.InputDialog* method), 146  
`winfo_atomname()` (*robot.libraries.dialogs\_py.MessageDialog* method), 132  
`winfo_atomname()` (*robot.libraries.dialogs\_py MultipleSelectionDialog* method), 174  
`winfo_atomname()` (*robot.libraries.dialogs\_py.PassFailDialog* method), 188  
`winfo_atomname()` (*robot.libraries.dialogs\_py.SelectionDialog* method), 160  
`winfo_cells()` (*robot.libraries.dialogs\_py.InputDialog* method), 146  
`winfo_cells()` (*robot.libraries.dialogs\_py.MessageDialog* method), 132  
`winfo_cells()` (*robot.libraries.dialogs\_py MultipleSelectionDialog* method), 174  
`winfo_cells()` (*robot.libraries.dialogs\_py.PassFailDialog* method), 188  
`winfo_cells()` (*robot.libraries.dialogs\_py.SelectionDialog* method), 160

`winfo_children()` (`robot.libraries.dialogs_py.InputDialog` method), 146

`winfo_children()` (`robot.libraries.dialogs_py.MessageDialog` method), 132

`winfo_children()` (`robot.libraries.dialogs_py.MultipleSelectionDialog` method), 174

`winfo_children()` (`robot.libraries.dialogs_py.PassFailDialog` method), 188

`winfo_children()` (`robot.libraries.dialogs_py.SelectionDialog` method), 160

`winfo_class()` (`robot.libraries.dialogs_py.InputDialog` method), 146

`winfo_class()` (`robot.libraries.dialogs_py.MessageDialog` method), 132

`winfo_class()` (`robot.libraries.dialogs_py.MultipleSelectionDialog` method), 174

`winfo_class()` (`robot.libraries.dialogs_py.PassFailDialog` method), 188

`winfo_class()` (`robot.libraries.dialogs_py.SelectionDialog` method), 160

`winfo_colormapfull()` (`robot.libraries.dialogs_py.InputDialog` method), 146

`winfo_colormapfull()` (`robot.libraries.dialogs_py.MessageDialog` method), 132

`winfo_colormapfull()` (`robot.libraries.dialogs_py.MultipleSelectionDialog` method), 174

`winfo_colormapfull()` (`robot.libraries.dialogs_py.PassFailDialog` method), 188

`winfo_colormapfull()` (`robot.libraries.dialogs_py.SelectionDialog` method), 160

`winfo_containing()` (`robot.libraries.dialogs_py.InputDialog` method), 146

`winfo_containing()` (`robot.libraries.dialogs_py.MessageDialog` method), 132

`winfo_containing()` (`robot.libraries.dialogs_py.MultipleSelectionDialog` method), 174

`winfo_containing()` (`robot.libraries.dialogs_py.PassFailDialog` method), 188

`winfo_containing()` (`robot.libraries.dialogs_py.SelectionDialog` method), 160

`winfo_depth()` (`robot.libraries.dialogs_py.InputDialog` method), 146

`winfo_depth()` (`robot.libraries.dialogs_py.MessageDialog` method), 132

`winfo_depth()` (`robot.libraries.dialogs_py.MultipleSelectionDialog` method), 174

`winfo_depth()` (`robot.libraries.dialogs_py.PassFailDialog` method), 188

`winfo_depth()` (`robot.libraries.dialogs_py.SelectionDialog` method), 160

`winfo_exists()` (`robot.libraries.dialogs_py.InputDialog` method), 146

`winfo_exists()` (`robot.libraries.dialogs_py.MessageDialog` method), 132

`winfo_exists()` (`robot.libraries.dialogs_py.MultipleSelectionDialog` method), 174

`winfo_exists()` (`robot.libraries.dialogs_py.PassFailDialog` method), 188

`winfo_exists()` (`robot.libraries.dialogs_py.SelectionDialog` method), 160

`winfo_fpixels()` (`robot.libraries.dialogs_py.InputDialog` method), 146

`winfo_fpixels()` (`robot.libraries.dialogs_py.MessageDialog` method), 132

`winfo_fpixels()` (`robot.libraries.dialogs_py.MultipleSelectionDialog` method), 174

`winfo_fpixels()` (`robot.libraries.dialogs_py.PassFailDialog` method), 188

`winfo_fpixels()` (`robot.libraries.dialogs_py.SelectionDialog` method), 160

`winfo_geometry()` (`robot.libraries.dialogs_py.InputDialog` method), 146

`winfo_geometry()` (`robot.libraries.dialogs_py.MessageDialog` method), 132

`winfo_geometry()` (`robot.libraries.dialogs_py.MultipleSelectionDialog` method), 174

`winfo_geometry()` (`robot.libraries.dialogs_py.PassFailDialog` method), 188

`winfo_geometry()` (`robot.libraries.dialogs_py.SelectionDialog` method), 160

`winfo_height()` (`robot.libraries.dialogs_py.InputDialog` method), 146

`winfo_height()` (`robot.libraries.dialogs_py.MessageDialog` method), 132

`winfo_height()` (`robot.libraries.dialogs_py.MultipleSelectionDialog` method), 174

`winfo_height()` (`robot.libraries.dialogs_py.PassFailDialog` method), 188

`winfo_height()` (`robot.libraries.dialogs_py.SelectionDialog` method), 160

`winfo_id()` (`robot.libraries.dialogs_py.InputDialog` method), 146

`winfo_id()` (`robot.libraries.dialogs_py.MessageDialog` method), 132

`winfo_id()` (`robot.libraries.dialogs_py.MultipleSelectionDialog` method), 174

`winfo_id()` (`robot.libraries.dialogs_py.PassFailDialog` method), 188

`winfo_id()` (`robot.libraries.dialogs_py.SelectionDialog` method), 160



`winfo_id()` (`robot.libraries.dialogs_py.SelectionDialog` `winfo_pathname()` (`robot.libraries.dialogs_py.MessageDialog`  
`method`), 160 `method`), 133  
`winfo_interps()` (`robot.libraries.dialogs_py.InputDialog` `winfo_pathname()` (`robot.libraries.dialogs_py.MultipleSelectionDialog`  
`method`), 146 `method`), 174  
`winfo_interps()` (`robot.libraries.dialogs_py.MessageDialog` `winfo_pathname()` (`robot.libraries.dialogs_py.PassFailDialog`  
`method`), 132 `method`), 188  
`winfo_interps()` (`robot.libraries.dialogs_py.MultipleSelectionDialog` `winfo_pathname()` (`robot.libraries.dialogs_py.SelectionDialog`  
`method`), 174 `method`), 160  
`winfo_interps()` (`robot.libraries.dialogs_py.PassFailDialog` `winfo_pixels()` (`robot.libraries.dialogs_py.InputDialog`  
`method`), 188 `method`), 146  
`winfo_interps()` (`robot.libraries.dialogs_py.SelectionDialog` `winfo_pixels()` (`robot.libraries.dialogs_py.MessageDialog`  
`method`), 160 `method`), 133  
`winfo_ismapped()` (`robot.libraries.dialogs_py.InputDialog` `winfo_pixels()` (`robot.libraries.dialogs_py.MultipleSelectionDialog`  
`method`), 146 `method`), 174  
`winfo_ismapped()` (`robot.libraries.dialogs_py.MessageDialog` `winfo_pixels()` (`robot.libraries.dialogs_py.PassFailDialog`  
`method`), 132 `method`), 188  
`winfo_ismapped()` (`robot.libraries.dialogs_py.MultipleSelectionDialog` `winfo_pixels()` (`robot.libraries.dialogs_py.SelectionDialog`  
`method`), 174 `method`), 160  
`winfo_ismapped()` (`robot.libraries.dialogs_py.PassFailDialog` `winfo_pointerx()` (`robot.libraries.dialogs_py.InputDialog`  
`method`), 188 `method`), 147  
`winfo_ismapped()` (`robot.libraries.dialogs_py.SelectionDialog` `winfo_pointerx()` (`robot.libraries.dialogs_py.MessageDialog`  
`method`), 160 `method`), 133  
`winfo_manager()` (`robot.libraries.dialogs_py.InputDialog` `winfo_pointerx()` (`robot.libraries.dialogs_py.MultipleSelectionDialog`  
`method`), 146 `method`), 175  
`winfo_manager()` (`robot.libraries.dialogs_py.MessageDialog` `winfo_pointerx()` (`robot.libraries.dialogs_py.PassFailDialog`  
`method`), 132 `method`), 189  
`winfo_manager()` (`robot.libraries.dialogs_py.MultipleSelectionDialog` `winfo_pointerx()` (`robot.libraries.dialogs_py.SelectionDialog`  
`method`), 174 `method`), 161  
`winfo_manager()` (`robot.libraries.dialogs_py.PassFailDialog` `winfo_pointerxy()`  
`method`), 188 (`robot.libraries.dialogs_py.InputDialog`  
`method`), 147  
`winfo_manager()` (`robot.libraries.dialogs_py.SelectionDialog` `winfo_pointerxy()`  
`method`), 160 (`robot.libraries.dialogs_py.MessageDialog`  
`method`), 133  
`winfo_name()` (`robot.libraries.dialogs_py.InputDialog` `winfo_pointerxy()`  
`method`), 146 (`robot.libraries.dialogs_py.MultipleSelectionDialog`  
`method`), 175  
`winfo_name()` (`robot.libraries.dialogs_py.MessageDialog` `winfo_pointerxy()`  
`method`), 132 (`robot.libraries.dialogs_py.MultipleSelectionDialog`  
`method`), 174  
`winfo_name()` (`robot.libraries.dialogs_py.MultipleSelectionDialog` `winfo_pointerxy()`  
`method`), 174 (`robot.libraries.dialogs_py.PassFailDialog`  
`method`), 189  
`winfo_name()` (`robot.libraries.dialogs_py.PassFailDialog` `winfo_pointerxy()`  
`method`), 188 (`robot.libraries.dialogs_py.PassFailDialog`  
`method`), 189  
`winfo_name()` (`robot.libraries.dialogs_py.SelectionDialog` `winfo_pointerxy()`  
`method`), 160 (`robot.libraries.dialogs_py.SelectionDialog`  
`method`), 161  
`winfo_parent()` (`robot.libraries.dialogs_py.InputDialog` `winfo_pointery()` (`robot.libraries.dialogs_py.InputDialog`  
`method`), 146 `method`), 147  
`winfo_parent()` (`robot.libraries.dialogs_py.MessageDialog` `winfo_pointery()` (`robot.libraries.dialogs_py.MessageDialog`  
`method`), 132 `method`), 133  
`winfo_parent()` (`robot.libraries.dialogs_py.MultipleSelectionDialog` `winfo_pointery()` (`robot.libraries.dialogs_py.MultipleSelectionDialog`  
`method`), 174 `method`), 175  
`winfo_parent()` (`robot.libraries.dialogs_py.PassFailDialog` `winfo_pointery()` (`robot.libraries.dialogs_py.PassFailDialog`  
`method`), 188 `method`), 189  
`winfo_parent()` (`robot.libraries.dialogs_py.SelectionDialog` `winfo_pointery()` (`robot.libraries.dialogs_py.SelectionDialog`  
`method`), 160 `method`), 161  
`winfo_pathname()` (`robot.libraries.dialogs_py.InputDialog` `winfo_reqheight()`  
`method`), 146

<code>(robot.libraries.dialogs_py.InputDialog</code>	<code>winfo_screen()</code> ( <code>robot.libraries.dialogs_py.InputDialog</code>
<code>method), 147</code>	<code>method), 147</code>
<code>winfo_reqheight()</code>	<code>winfo_screen()</code> ( <code>robot.libraries.dialogs_py.MessageDialog</code>
<code>(robot.libraries.dialogs_py.MessageDialog</code>	<code>method), 133</code>
<code>method), 133</code>	<code>winfo_screen()</code> ( <code>robot.libraries.dialogs_pyMultipleSelectionDialog</code>
<code>winfo_reqheight()</code>	<code>method), 175</code>
<code>(robot.libraries.dialogs_pyMultipleSelectionDialog</code>	<code>winfo_screen()</code> ( <code>robot.libraries.dialogs_py.PassFailDialog</code>
<code>method), 175</code>	<code>method), 189</code>
<code>winfo_reqheight()</code>	<code>winfo_screen()</code> ( <code>robot.libraries.dialogs_py.SelectionDialog</code>
<code>(robot.libraries.dialogs_py.PassFailDialog</code>	<code>method), 161</code>
<code>method), 189</code>	<code>winfo_screencells()</code>
<code>winfo_reqheight()</code>	<code>(robot.libraries.dialogs_py.InputDialog</code>
<code>(robot.libraries.dialogs_py.SelectionDialog</code>	<code>method), 147</code>
<code>method), 161</code>	<code>winfo_screencells()</code>
<code>winfo_reqwidth()</code> ( <code>robot.libraries.dialogs_py.InputDialog</code>	<code>(robot.libraries.dialogs_py.MessageDialog</code>
<code>method), 147</code>	<code>method), 133</code>
<code>winfo_reqwidth()</code> ( <code>robot.libraries.dialogs_py.MessageDialog</code>	<code>winfo_screencells()</code>
<code>method), 133</code>	<code>(robot.libraries.dialogs_pyMultipleSelectionDialog</code>
<code>winfo_reqwidth()</code> ( <code>robot.libraries.dialogs_pyMultipleSelectionDialog</code>	<code>method), 175</code>
<code>method), 175</code>	<code>winfo_screencells()</code>
<code>winfo_reqwidth()</code> ( <code>robot.libraries.dialogs_py.PassFailDialog</code>	<code>(robot.libraries.dialogs_py.PassFailDialog</code>
<code>method), 189</code>	<code>method), 189</code>
<code>winfo_reqwidth()</code> ( <code>robot.libraries.dialogs_py.SelectionDialog</code>	<code>winfo_screencells()</code>
<code>method), 161</code>	<code>(robot.libraries.dialogs_py.SelectionDialog</code>
<code>winfo_rgb()</code> ( <code>robot.libraries.dialogs_py.InputDialog</code>	<code>method), 161</code>
<code>method), 147</code>	<code>winfo_screendepth()</code>
<code>winfo_rgb()</code> ( <code>robot.libraries.dialogs_py.MessageDialog</code>	<code>(robot.libraries.dialogs_py.InputDialog</code>
<code>method), 133</code>	<code>method), 147</code>
<code>winfo_rgb()</code> ( <code>robot.libraries.dialogs_pyMultipleSelectionDialog</code>	<code>winfo_screendepth()</code>
<code>method), 175</code>	<code>(robot.libraries.dialogs_py.MessageDialog</code>
<code>winfo_rgb()</code> ( <code>robot.libraries.dialogs_py.PassFailDialog</code>	<code>method), 133</code>
<code>method), 189</code>	<code>winfo_screendepth()</code>
<code>winfo_rgb()</code> ( <code>robot.libraries.dialogs_py.SelectionDialog</code>	<code>(robot.libraries.dialogs_pyMultipleSelectionDialog</code>
<code>method), 161</code>	<code>method), 175</code>
<code>winfo_rootx()</code> ( <code>robot.libraries.dialogs_py.InputDialog</code>	<code>winfo_screendepth()</code>
<code>method), 147</code>	<code>(robot.libraries.dialogs_py.PassFailDialog</code>
<code>winfo_rootx()</code> ( <code>robot.libraries.dialogs_py.MessageDialog</code>	<code>method), 189</code>
<code>method), 133</code>	<code>winfo_screendepth()</code>
<code>winfo_rootx()</code> ( <code>robot.libraries.dialogs_pyMultipleSelectionDialog</code>	<code>(robot.libraries.dialogs_py.SelectionDialog</code>
<code>method), 175</code>	<code>method), 161</code>
<code>winfo_rootx()</code> ( <code>robot.libraries.dialogs_py.PassFailDialog</code>	<code>winfo_screenheight()</code>
<code>method), 189</code>	<code>(robot.libraries.dialogs_py.InputDialog</code>
<code>winfo_rootx()</code> ( <code>robot.libraries.dialogs_py.SelectionDialog</code>	<code>method), 147</code>
<code>method), 161</code>	<code>winfo_screenheight()</code>
<code>winfo_rooty()</code> ( <code>robot.libraries.dialogs_py.InputDialog</code>	<code>(robot.libraries.dialogs_py.MessageDialog</code>
<code>method), 147</code>	<code>method), 133</code>
<code>winfo_rooty()</code> ( <code>robot.libraries.dialogs_py.MessageDialog</code>	<code>winfo_screenheight()</code>
<code>method), 133</code>	<code>(robot.libraries.dialogs_pyMultipleSelectionDialog</code>
<code>winfo_rooty()</code> ( <code>robot.libraries.dialogs_pyMultipleSelectionDialog</code>	<code>method), 175</code>
<code>method), 175</code>	<code>winfo_screenheight()</code>
<code>winfo_rooty()</code> ( <code>robot.libraries.dialogs_py.PassFailDialog</code>	<code>(robot.libraries.dialogs_py.PassFailDialog</code>
<code>method), 189</code>	<code>method), 189</code>
<code>winfo_rooty()</code> ( <code>robot.libraries.dialogs_py.SelectionDialog</code>	<code>winfo_screenheight()</code>
<code>method), 161</code>	<code>(robot.libraries.dialogs_py.SelectionDialog</code>

`method`), 161  
`winfo_screenmmheight()`  
     (`robot.libraries.dialogs_py.InputDialog`  
     `method`), 147  
`winfo_screenmmheight()`  
     (`robot.libraries.dialogs_py.MessageDialog`  
     `method`), 133  
`winfo_screenmmheight()`  
     (`robot.libraries.dialogs_py.MultipleSelectionDialog`  
     `method`), 175  
`winfo_screenmmheight()`  
     (`robot.libraries.dialogs_py.PassFailDialog`  
     `method`), 189  
`winfo_screenmmheight()`  
     (`robot.libraries.dialogs_py.SelectionDialog`  
     `method`), 161  
`winfo_screenmmwidth()`  
     (`robot.libraries.dialogs_py.InputDialog`  
     `method`), 147  
`winfo_screenmmwidth()`  
     (`robot.libraries.dialogs_py.MessageDialog`  
     `method`), 133  
`winfo_screenmmwidth()`  
     (`robot.libraries.dialogs_py.MultipleSelectionDialog`  
     `method`), 175  
`winfo_screenmmwidth()`  
     (`robot.libraries.dialogs_py.PassFailDialog`  
     `method`), 189  
`winfo_screenmmwidth()`  
     (`robot.libraries.dialogs_py.SelectionDialog`  
     `method`), 161  
`winfo_screenvisual()`  
     (`robot.libraries.dialogs_py.InputDialog`  
     `method`), 147  
`winfo_screenvisual()`  
     (`robot.libraries.dialogs_py.MessageDialog`  
     `method`), 133  
`winfo_screenvisual()`  
     (`robot.libraries.dialogs_py.MultipleSelectionDialog`  
     `method`), 175  
`winfo_screenvisual()`  
     (`robot.libraries.dialogs_py.PassFailDialog`  
     `method`), 189  
`winfo_screenvisual()`  
     (`robot.libraries.dialogs_py.SelectionDialog`  
     `method`), 161  
`winfo_screenwidth()`  
     (`robot.libraries.dialogs_py.InputDialog`  
     `method`), 147  
`winfo_screenwidth()`  
     (`robot.libraries.dialogs_py.MessageDialog`  
     `method`), 133  
`winfo_screenwidth()`  
     (`robot.libraries.dialogs_py.MultipleSelectionDialog`  
     `method`), 175  
`winfo_screenwidth()`  
     (`robot.libraries.dialogs_py.PassFailDialog`  
     `method`), 189  
`winfo_screenwidth()`  
     (`robot.libraries.dialogs_py.SelectionDialog`  
     `method`), 161  
`winfo_server()` (`robot.libraries.dialogs_py.InputDialog`  
     `method`), 147  
`winfo_server()` (`robot.libraries.dialogs_py.MessageDialog`  
     `method`), 133  
`winfo_server()` (`robot.libraries.dialogs_py.MultipleSelectionDialog`  
     `method`), 175  
`winfo_server()` (`robot.libraries.dialogs_py.PassFailDialog`  
     `method`), 189  
`winfo_server()` (`robot.libraries.dialogs_py.SelectionDialog`  
     `method`), 161  
`winfo_toplevel()` (`robot.libraries.dialogs_py.InputDialog`  
     `method`), 147  
`winfo_toplevel()` (`robot.libraries.dialogs_py.MessageDialog`  
     `method`), 133  
`winfo_toplevel()` (`robot.libraries.dialogs_py.MultipleSelectionDialog`  
     `method`), 175  
`winfo_toplevel()` (`robot.libraries.dialogs_py.PassFailDialog`  
     `method`), 189  
`winfo_toplevel()` (`robot.libraries.dialogs_py.SelectionDialog`  
     `method`), 161  
`winfo_viewable()` (`robot.libraries.dialogs_py.InputDialog`  
     `method`), 147  
`winfo_viewable()` (`robot.libraries.dialogs_py.MessageDialog`  
     `method`), 133  
`winfo_viewable()` (`robot.libraries.dialogs_py.MultipleSelectionDialog`  
     `method`), 175  
`winfo_viewable()` (`robot.libraries.dialogs_py.PassFailDialog`  
     `method`), 189  
`winfo_viewable()` (`robot.libraries.dialogs_py.SelectionDialog`  
     `method`), 161  
`winfo_visual()` (`robot.libraries.dialogs_py.InputDialog`  
     `method`), 147  
`winfo_visual()` (`robot.libraries.dialogs_py.MessageDialog`  
     `method`), 134  
`winfo_visual()` (`robot.libraries.dialogs_py.MultipleSelectionDialog`  
     `method`), 175  
`winfo_visual()` (`robot.libraries.dialogs_py.PassFailDialog`  
     `method`), 189  
`winfo_visual()` (`robot.libraries.dialogs_py.SelectionDialog`  
     `method`), 161  
`winfo_visualid()` (`robot.libraries.dialogs_py.InputDialog`  
     `method`), 148  
`winfo_visualid()` (`robot.libraries.dialogs_py.MessageDialog`  
     `method`), 134  
`winfo_visualid()` (`robot.libraries.dialogs_py.MultipleSelectionDialog`  
     `method`), 176  
`winfo_visualid()` (`robot.libraries.dialogs_py.PassFailDialog`  
     `method`), 189  
`winfo_visualid()` (`robot.libraries.dialogs_py.SelectionDialog`  
     `method`), 161

*method*), 190

`winfo_visualid()` (*robot.libraries.dialogs\_py.SelectionDialog* *method*), 190

*method*), 162

`winfo_visualsavailable()`

    (*robot.libraries.dialogs\_py.InputDialog* *method*), 148

`winfo_visualsavailable()`

    (*robot.libraries.dialogs\_py.MessageDialog* *method*), 134

`winfo_visualsavailable()`

    (*robot.libraries.dialogs\_py.MultipleSelectionDialog* *method*), 176

`winfo_visualsavailable()`

    (*robot.libraries.dialogs\_py.PassFailDialog* *method*), 190

`winfo_visualsavailable()`

    (*robot.libraries.dialogs\_py.SelectionDialog* *method*), 162

`winfo_vrootheight()`

    (*robot.libraries.dialogs\_py.InputDialog* *method*), 148

`winfo_vrootheight()`

    (*robot.libraries.dialogs\_py.MessageDialog* *method*), 134

`winfo_vrootheight()`

    (*robot.libraries.dialogs\_py.MultipleSelectionDialog* *method*), 176

`winfo_vrootheight()`

    (*robot.libraries.dialogs\_py.PassFailDialog* *method*), 190

`winfo_vrootheight()`

    (*robot.libraries.dialogs\_py.SelectionDialog* *method*), 162

`winfo_vrootwidth()`

    (*robot.libraries.dialogs\_py.InputDialog* *method*), 148

`winfo_vrootwidth()`

    (*robot.libraries.dialogs\_py.MessageDialog* *method*), 134

`winfo_vrootwidth()`

    (*robot.libraries.dialogs\_py.MultipleSelectionDialog* *method*), 176

`winfo_vrootwidth()`

    (*robot.libraries.dialogs\_py.PassFailDialog* *method*), 190

`winfo_vrootwidth()`

    (*robot.libraries.dialogs\_py.SelectionDialog* *method*), 162

`winfo_vrootx()` (*robot.libraries.dialogs\_py.InputDialog* *method*), 148

`winfo_vrootx()` (*robot.libraries.dialogs\_py.MessageDialog* *method*), 134

`winfo_vrootx()` (*robot.libraries.dialogs\_py.MultipleSelectionDialog* *method*), 176

`winfo_vrootx()` (*robot.libraries.dialogs\_py.PassFailDialog* *method*), 190

`winfo_vrootx()` (*robot.libraries.dialogs\_py.SelectionDialog* *method*), 162

`winfo_vrootx()` (*robot.libraries.dialogs\_py.PassFailDialog* *method*), 190

`winfo_vrootx()` (*robot.libraries.dialogs\_py.SelectionDialog* *method*), 162

`winfo_vrootx()` (*robot.libraries.dialogs\_py.MessageDialog* *method*), 134

`winfo_vrootx()` (*robot.libraries.dialogs\_py.MultipleSelectionDialog* *method*), 176

`winfo_vrootx()` (*robot.libraries.dialogs\_py.PassFailDialog* *method*), 190

`winfo_vrootx()` (*robot.libraries.dialogs\_py.SelectionDialog* *method*), 162

`winfo_vrootx()` (*robot.libraries.dialogs\_py.PassFailDialog* *method*), 190

`winfo_vrootx()` (*robot.libraries.dialogs\_py.SelectionDialog* *method*), 162

`winfo_vrootx()` (*robot.libraries.dialogs\_py.InputDialog* *method*), 148

`winfo_vrootx()` (*robot.libraries.dialogs\_py.MessageDialog* *method*), 134

`winfo_vrootx()` (*robot.libraries.dialogs\_py.MultipleSelectionDialog* *method*), 176

`winfo_vrootx()` (*robot.libraries.dialogs\_py.PassFailDialog* *method*), 190

`winfo_vrootx()` (*robot.libraries.dialogs\_py.SelectionDialog* *method*), 162

`WITH_NAME` (*robot.parsing.lexer.tokens.END* *attribute*), 305

`WITH_NAME` (*robot.parsing.lexer.tokens.EOS* *attribute*), 303

`WITH_NAME` (*robot.parsing.lexer.tokens.Token* *attribute*), 300

`with_traceback()` (*robot.api.exceptions.ContinuableFailure* *method*), 13

`with_traceback()` (*robot.api.exceptions.Error* *method*), 13



<code>with_traceback()</code> ( <i>robot.api.exceptions.Failure</i> method), 12	<code>withdraw()</code> ( <i>robot.libraries.dialogs_py.SelectionDialog</i> method), 162
<code>with_traceback()</code> ( <i>robot.api.exceptions.FatalError</i> method), 13	<code>wm_aspect()</code> ( <i>robot.libraries.dialogs_py.InputDialog</i> method), 148
<code>with_traceback()</code> ( <i>robot.api.exceptions.SkipExecution</i> method), 14	<code>wm_aspect()</code> ( <i>robot.libraries.dialogs_py.MessageDialog</i> method), 134
<code>with_traceback()</code> ( <i>robot.errors.BreakLoop</i> method), 579	<code>wm_aspect()</code> ( <i>robot.libraries.dialogs_py.MultipleSelectionDialog</i> method), 176
<code>with_traceback()</code> ( <i>robot.errors.ContinueLoop</i> method), 579	<code>wm_aspect()</code> ( <i>robot.libraries.dialogs_py.PassFailDialog</i> method), 190
<code>with_traceback()</code> ( <i>robot.errors.DataError</i> method), 575	<code>wm_aspect()</code> ( <i>robot.libraries.dialogs_py.SelectionDialog</i> method), 162
<code>with_traceback()</code> ( <i>robot.errors.ExecutionFailed</i> method), 576	<code>wm_attributes()</code> ( <i>robot.libraries.dialogs_py.InputDialog</i> method), 148
<code>with_traceback()</code> ( <i>robot.errors.ExecutionFailures</i> method), 577	<code>wm_attributes()</code> ( <i>robot.libraries.dialogs_py.MessageDialog</i> method), 134
<code>with_traceback()</code> ( <i>robot.errors.ExecutionPassed</i> method), 578	<code>wm_attributes()</code> ( <i>robot.libraries.dialogs_py.MultipleSelectionDialog</i> method), 176
<code>with_traceback()</code> ( <i>robot.errors.ExecutionStatus</i> method), 576	<code>wm_attributes()</code> ( <i>robot.libraries.dialogs_py.PassFailDialog</i> method), 190
<code>with_traceback()</code> ( <i>robot.errors.FrameworkError</i> method), 575	<code>wm_attributes()</code> ( <i>robot.libraries.dialogs_py.SelectionDialog</i> method), 162
<code>with_traceback()</code> ( <i>robot.errors.HandlerExecutionFailed</i> method), 577	<code>wm_client()</code> ( <i>robot.libraries.dialogs_py.InputDialog</i> method), 148
<code>with_traceback()</code> ( <i>robot.errors.Information</i> method), 576	<code>wm_client()</code> ( <i>robot.libraries.dialogs_py.MessageDialog</i> method), 134
<code>with_traceback()</code> ( <i>robot.errors.KeywordError</i> method), 575	<code>wm_client()</code> ( <i>robot.libraries.dialogs_py.MultipleSelectionDialog</i> method), 176
<code>with_traceback()</code> ( <i>robot.errors.PassExecution</i> method), 578	<code>wm_client()</code> ( <i>robot.libraries.dialogs_py.PassFailDialog</i> method), 190
<code>with_traceback()</code> ( <i>robot.errors.RemoteError</i> method), 580	<code>wm_client()</code> ( <i>robot.libraries.dialogs_py.SelectionDialog</i> method), 162
<code>with_traceback()</code> ( <i>robot.errors.ReturnFromKeyword</i> method), 579	<code>wm_colormapwindows()</code> ( <i>robot.libraries.dialogs_py.InputDialog</i> method), 148
<code>with_traceback()</code> ( <i>robot.errors.RobotError</i> method), 574	<code>wm_colormapwindows()</code> ( <i>robot.libraries.dialogs_py.MessageDialog</i> method), 135
<code>with_traceback()</code> ( <i>robot.errors.TimeoutError</i> method), 576	<code>wm_colormapwindows()</code> ( <i>robot.libraries.dialogs_py.MultipleSelectionDialog</i> method), 176
<code>with_traceback()</code> ( <i>robot.errors.UserKeywordExecutionFailed</i> method), 577	<code>wm_colormapwindows()</code> ( <i>robot.libraries.dialogs_py.PassFailDialog</i> method), 190
<code>with_traceback()</code> ( <i>robot.errors.VariableError</i> method), 575	<code>wm_colormapwindows()</code> ( <i>robot.libraries.dialogs_py.SelectionDialog</i> method), 162
<code>with_traceback()</code> ( <i>robot.libraries.BuiltIn.RobotNotRunningError</i> method), 65	<code>wm_command()</code> ( <i>robot.libraries.dialogs_py.InputDialog</i> method), 149
<code>with_traceback()</code> ( <i>robot.libraries.Telnet.NoMatchError</i> method), 112	<code>wm_command()</code> ( <i>robot.libraries.dialogs_py.MessageDialog</i> method), 135
<code>withdraw()</code> ( <i>robot.libraries.dialogs_py.InputDialog</i> method), 148	<code>wm_command()</code> ( <i>robot.libraries.dialogs_py.MultipleSelectionDialog</i> method), 177
<code>withdraw()</code> ( <i>robot.libraries.dialogs_py.MessageDialog</i> method), 134	<code>wm_command()</code> ( <i>robot.libraries.dialogs_py.PassFailDialog</i> method), 190
<code>withdraw()</code> ( <i>robot.libraries.dialogs_py.MultipleSelectionDialog</i> method), 176	
<code>withdraw()</code> ( <i>robot.libraries.dialogs_py.PassFailDialog</i> method), 190	

*method*), 191

`wm_command()` (*robot.libraries.dialogs\_py.SelectionDialog* *method*), 163

`wm_deiconify()` (*robot.libraries.dialogs\_py.InputDialog* *method*), 149

`wm_deiconify()` (*robot.libraries.dialogs\_py.MessageDialog* *method*), 135

`wm_deiconify()` (*robot.libraries.dialogs\_py.MultipleSelectionDialog* *method*), 177

`wm_deiconify()` (*robot.libraries.dialogs\_py.PassFailDialog* *method*), 191

`wm_deiconify()` (*robot.libraries.dialogs\_py.SelectionDialog* *method*), 163

`wm_focusmodel()` (*robot.libraries.dialogs\_py.InputDialog* *method*), 149

`wm_focusmodel()` (*robot.libraries.dialogs\_py.MessageDialog* *method*), 135

`wm_focusmodel()` (*robot.libraries.dialogs\_py.MultipleSelectionDialog* *method*), 177

`wm_focusmodel()` (*robot.libraries.dialogs\_py.PassFailDialog* *method*), 191

`wm_focusmodel()` (*robot.libraries.dialogs\_py.SelectionDialog* *method*), 163

`wm_forget()` (*robot.libraries.dialogs\_py.InputDialog* *method*), 149

`wm_forget()` (*robot.libraries.dialogs\_py.MessageDialog* *method*), 135

`wm_forget()` (*robot.libraries.dialogs\_py.MultipleSelectionDialog* *method*), 177

`wm_forget()` (*robot.libraries.dialogs\_py.PassFailDialog* *method*), 191

`wm_forget()` (*robot.libraries.dialogs\_py.SelectionDialog* *method*), 163

`wm_frame()` (*robot.libraries.dialogs\_py.InputDialog* *method*), 149

`wm_frame()` (*robot.libraries.dialogs\_py.MessageDialog* *method*), 135

`wm_frame()` (*robot.libraries.dialogs\_py.MultipleSelectionDialog* *method*), 177

`wm_frame()` (*robot.libraries.dialogs\_py.PassFailDialog* *method*), 191

`wm_frame()` (*robot.libraries.dialogs\_py.SelectionDialog* *method*), 163

`wm_geometry()` (*robot.libraries.dialogs\_py.InputDialog* *method*), 149

`wm_geometry()` (*robot.libraries.dialogs\_py.MessageDialog* *method*), 135

`wm_geometry()` (*robot.libraries.dialogs\_py.MultipleSelectionDialog* *method*), 177

`wm_geometry()` (*robot.libraries.dialogs\_py.PassFailDialog* *method*), 191

`wm_geometry()` (*robot.libraries.dialogs\_py.SelectionDialog* *method*), 163

`wm_grid()` (*robot.libraries.dialogs\_py.InputDialog* *method*), 149

`wm_grid()` (*robot.libraries.dialogs\_py.MessageDialog* *method*), 135

`wm_grid()` (*robot.libraries.dialogs\_py.MultipleSelectionDialog* *method*), 177

`wm_grid()` (*robot.libraries.dialogs\_py.PassFailDialog* *method*), 191

`wm_grid()` (*robot.libraries.dialogs\_py.SelectionDialog* *method*), 163

`wm_group()` (*robot.libraries.dialogs\_py.InputDialog* *method*), 149

`wm_group()` (*robot.libraries.dialogs\_py.MessageDialog* *method*), 135

`wm_group()` (*robot.libraries.dialogs\_py.MultipleSelectionDialog* *method*), 177

`wm_group()` (*robot.libraries.dialogs\_py.PassFailDialog* *method*), 191

`wm_group()` (*robot.libraries.dialogs\_py.SelectionDialog* *method*), 163

`wm_iconbitmap()` (*robot.libraries.dialogs\_py.InputDialog* *method*), 149

`wm_iconbitmap()` (*robot.libraries.dialogs\_py.MessageDialog* *method*), 135

`wm_iconbitmap()` (*robot.libraries.dialogs\_py.MultipleSelectionDialog* *method*), 177

`wm_iconbitmap()` (*robot.libraries.dialogs\_py.PassFailDialog* *method*), 191

`wm_iconbitmap()` (*robot.libraries.dialogs\_py.SelectionDialog* *method*), 163

`wm_iconify()` (*robot.libraries.dialogs\_py.InputDialog* *method*), 149

`wm_iconify()` (*robot.libraries.dialogs\_py.MessageDialog* *method*), 135

`wm_iconify()` (*robot.libraries.dialogs\_py.MultipleSelectionDialog* *method*), 177

`wm_iconify()` (*robot.libraries.dialogs\_py.PassFailDialog* *method*), 191

`wm_iconify()` (*robot.libraries.dialogs\_py.SelectionDialog* *method*), 163

`wm_iconmask()` (*robot.libraries.dialogs\_py.InputDialog* *method*), 149

`wm_iconmask()` (*robot.libraries.dialogs\_py.MessageDialog* *method*), 135

`wm_iconmask()` (*robot.libraries.dialogs\_py.MultipleSelectionDialog* *method*), 177

`wm_iconmask()` (*robot.libraries.dialogs\_py.PassFailDialog* *method*), 191

`wm_iconmask()` (*robot.libraries.dialogs\_py.SelectionDialog* *method*), 163

`wm_iconname()` (*robot.libraries.dialogs\_py.InputDialog* *method*), 149

`wm_iconname()` (*robot.libraries.dialogs\_py.MessageDialog* *method*), 135

`wm_iconname()` (*robot.libraries.dialogs\_py.MultipleSelectionDialog* *method*), 177

`wm_iconname()` (*robot.libraries.dialogs\_py.PassFailDialog* *method*), 191

`wm_iconname()` (*robot.libraries.dialogs\_py.SelectionDialog* *method*), 163

755

`wm_protocol()` (`robot.libraries.dialogs_py.SelectionDialog` `withdraw()` (`robot.libraries.dialogs_py.MessageDialog` `method`), 164 `method`), 136

`wm_resizable()` (`robot.libraries.dialogs_py.InputDialog` `wm_withdraw()` (`robot.libraries.dialogs_py.MultipleSelectionDialog` `method`), 150 `method`), 178

`wm_resizable()` (`robot.libraries.dialogs_py.MessageDialog` `withdraw()` (`robot.libraries.dialogs_py.PassFailDialog` `method`), 136 `method`), 192

`wm_resizable()` (`robot.libraries.dialogs_py.MultipleSelectionDialog` `withdraw()` (`robot.libraries.dialogs_py.SelectionDialog` `method`), 178 `method`), 164

`wm_resizable()` (`robot.libraries.dialogs_py.PassFailDialog` `write()` (in module `robot.api.logger`), 14 `method`), 192 `write()` (in module `robot.output.librarylogger`), 270

`wm_resizable()` (`robot.libraries.dialogs_py.SelectionDialog` `write()` (`robot.htmldata.htmlfilewriter.CssFileWriter` `method`), 164 `method`), 34

`wm_sizefrom()` (`robot.libraries.dialogs_py.InputDialog` `write()` (`robot.htmldata.htmlfilewriter.GeneratorWriter` `method`), 150 `method`), 34

`wm_sizefrom()` (`robot.libraries.dialogs_py.MessageDialog` `write()` (`robot.htmldata.htmlfilewriter.HtmlFileWriter` `method`), 136 `method`), 33

`wm_sizefrom()` (`robot.libraries.dialogs_py.MultipleSelectionDialog` `write()` (`robot.htmldata.htmlfilewriter.JsFileWriter` `method`), 178 `method`), 34

`wm_sizefrom()` (`robot.libraries.dialogs_py.PassFailDialog` `write()` (`robot.htmldata.htmlfilewriter.LineWriter` `method`), 192 `method`), 34

`wm_sizefrom()` (`robot.libraries.dialogs_py.SelectionDialog` `write()` (`robot.htmldata.htmlfilewriter.ModelWriter` `method`), 164 `method`), 34

`wm_state()` (`robot.libraries.dialogs_py.InputDialog` `write()` (`robot.htmldata.jsonwriter.JsonWriter` `method`), 150 `method`), 34

`wm_state()` (`robot.libraries.dialogs_py.MessageDialog` `write()` (`robot.libdocpkg.htmlwriter.LibdocHtmlWriter` `method`), 136 `method`), 37

`wm_state()` (`robot.libraries.dialogs_py.MultipleSelectionDialog` `write()` (`robot.libdocpkg.htmlwriter.LibdocModelWriter` `method`), 178 `method`), 37

`wm_state()` (`robot.libraries.dialogs_py.PassFailDialog` `write()` (`robot.libdocpkg.jsonwriter.LibdocJsonWriter` `method`), 192 `method`), 37

`wm_state()` (`robot.libraries.dialogs_py.SelectionDialog` `write()` (`robot.libdocpkg.xmlwriter.LibdocXmlWriter` `method`), 164 `method`), 39

`wm_title()` (`robot.libraries.dialogs_py.InputDialog` `write()` (`robot.libraries.Telnet.TelnetConnection` `method`), 150 `method`), 108

`wm_title()` (`robot.libraries.dialogs_py.MessageDialog` `write()` (`robot.output.console.highlighting.HighlightingStream` `method`), 136 `method`), 268

`wm_title()` (`robot.libraries.dialogs_py.MultipleSelectionDialog` `write()` (`robot.output.filelogger.FileLogger` `method`), 178 `method`), 270

`wm_title()` (`robot.libraries.dialogs_py.PassFailDialog` `write()` (`robot.output.logger.Logger` `method`), 192 `method`), 274

`wm_title()` (`robot.libraries.dialogs_py.SelectionDialog` `write()` (`robot.output.loggerhelper.AbstractLogger` `method`), 164 `method`), 274

`wm_title()` (`robot.libraries.dialogs_py.InputDialog` `write()` (`robot.output.output.Output` `method`), 150 `method`), 276

`wm_transient()` (`robot.libraries.dialogs_py.MessageDialog` `write()` (`robot.parsing.model.blocks.ModelWriter` `method`), 136 `method`), 310

`wm_transient()` (`robot.libraries.dialogs_py.MultipleSelectionDialog` `write()` (`robot.reporting.jswriter.JsResultWriter` `method`), 178 `method`), 359

`wm_transient()` (`robot.libraries.dialogs_py.PassFailDialog` `write()` (`robot.reporting.jswriter.SplitLogWriter` `method`), 192 `method`), 359

`wm_transient()` (`robot.libraries.dialogs_py.SelectionDialog` `write()` (`robot.reporting.jswriter.SuiteWriter` `method`), 164 `method`), 359

`wm_transient()` (`robot.libraries.dialogs_py.InputDialog` `write()` (`robot.reporting.logreportwriters.LogWriter` `method`), 150 `method`), 359

`wm_withdraw()` (`robot.libraries.dialogs_py.InputDialog` `write()` (`robot.reporting.logreportwriters.ReportWriter` `method`), 150 `method`), 359



[write\(\) \(robot.reporting.logreportwriters.RobotModelWriter method\), 359](#)  
[write\(\) \(robot.reporting.xunitwriter.XUnitWriter method\), 366](#)  
[write\(\) \(robot.testdoc.TestdocModelWriter method\), 585](#)  
[write\\_bare\(\) \(robot.libraries.Telnet.TelnetConnection method\), 108](#)  
[write\\_control\\_character\(\) \(robot.libraries.Telnet.TelnetConnection method\), 109](#)  
[write\\_data\(\) \(robot.testdoc.TestdocModelWriter method\), 585](#)  
[write\\_json\(\) \(robot.htmldata.jsonwriter.JsonWriter method\), 34](#)  
[write\\_results\(\) \(robot.reporting.resultwriter.ResultWriter method\), 365](#)  
[write\\_until\\_expected\\_output\(\) \(robot.libraries.Telnet.TelnetConnection method\), 108](#)

## X

[XML \(class in robot.libraries.XML\), 112](#)  
[xml\\_escape\(\) \(in module robot.utils.markuputils\), 564](#)  
[XmlDocBuilder \(class in robot.libdocpkg.xmlbuilder\), 38](#)  
[XmlElementHandler \(class in robot.result.xmlelementhandlers\), 483](#)  
[XmlLogger \(class in robot.output.xmllogger\), 278](#)  
[XmlRpcRemoteClient \(class in robot.libraries.Remote\), 94](#)  
[XmlWriter \(class in robot.utils.markupwriters\), 564](#)  
[xunit \(robot.conf.settings.RebotSettings attribute\), 33](#)  
[xunit \(robot.conf.settings.RobotSettings attribute\), 32](#)  
[XUnitFileWriter \(class in robot.reporting.xunitwriter\), 366](#)  
[XUnitWriter \(class in robot.reporting.xunitwriter\), 366](#)

## Y

[YamlImporter \(class in robot.variables.filesetter\), 569](#)  
[yellow\(\) \(robot.output.console.highlighting.AnsiHighlighter method\), 268](#)  
[yellow\(\) \(robot.output.console.highlighting.DosHighlighter method\), 269](#)  
[yellow\(\) \(robot.output.console.highlighting.NoHighlighting method\), 268](#)