Robot Framework Documentation

Release 6.0.2

Robot Framework developers

Jan 08, 2023

Contents

1 Entry points **3**

2 Public API **5**

3 All packages **7**
3.1 robot package 7

4 Indices **627**

Python Module Index **629**

Index **633**

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

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

CHAPTER 1

Entry points

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

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

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*.
- *Languages* and *Language* classes for external tools that need to work with different translations. The latter is also the base class to use with custom translations.

All of the above names can be imported like:

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

See documentations of the individual APIs for more details.

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

CHAPTER 3

All packages

All *robot* packages are listed below. Typically you should not need to import anything from them directly, but the above public APIs may return objects implemented in them.

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 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*.
- *Languages* and *Language* classes for external tools that need to work with different translations. The latter is also the base class to use with custom translations.

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)
```

(continues on next page)

(continued from previous page)

```
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*
- *Config* (new in 6.0)
- *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 the 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})")
```

(continues on next page)

(continued from previous page)

```
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 the 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):
```

(continues on next page)

(continued from previous page)

```
    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.'''
        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 updated 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):
```

(continues on next page)

(continued from previous page)

```
        # 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, '    '),
            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

```

(continues on next page)

(continued from previous page)

```
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.languages module

class robot.conf.languages.**Languages**(*languages=None*, *add_english=True*)
Bases: object

Keeps a list of languages and unifies the translations in the properties.

Example:

```
languages = Languages('de', add_english=False)
print(languages.settings)
languages = Languages(['pt-BR', 'Finnish', 'MyLang.py'])
for lang in languages:
    print(lang.name, lang.code)
```

Parameters

- **languages** – Initial language or list of languages. Languages can be given as language codes or names, paths or names of language modules to load, or as *Language* instances.
- **add_english** – If True, English is added automatically.

Raises *DataError* if a given language is not found.

add_language() can be used to add languages after initialization.

reset(*languages=None*, *add_english=True*)
Resets the instance to the given languages.

add_language(*lang*)
Add new language.

Parameters **lang** – Language to add. Can be a language code or name, name or path of a language module to load, or a *Language* instance.

Raises *DataError* if the language is not found.

Language codes and names are passed to by *Language.from_name()*. Language modules are imported and *Language* subclasses in them loaded.

class robot.conf.languages.**Language**
Bases: object

Base class for language definitions.

New translations can be added by extending this class and setting class attributes listed below.

Language *code* is got based on the class name and *name* based on the docstring.

settings_header = None

variables_header = None

test_cases_header = None

tasks_header = None

keywords_header = None

comments_header = None

library_setting = None

resource_setting = None

variables_setting = None

documentation_setting = None

metadata_setting = None

suite_setup_setting = None

suite_teardown_setting = None

test_setup_setting = None

task_setup_setting = None

test_teardown_setting = None

task_teardown_setting = None

test_template_setting = None

task_template_setting = None

test_timeout_setting = None

task_timeout_setting = None

test_tags_setting = None

task_tags_setting = None

keyword_tags_setting = None

tags_setting = None

setup_setting = None

teardown_setting = None

template_setting = None

timeout_setting = None

arguments_setting = None

given_prefixes = []

when_prefixes = []

then_prefixes = []

and_prefixes = []

but_prefixes = []

true_strings = []

false_strings = []

classmethod from_name(*name*)

Return language class based on given *name*.

Name can either be a language name (e.g. 'Finnish' or 'Brazilian Portuguese') or a language code (e.g. 'fi' or 'pt-BR'). Matching is case and space insensitive and the hyphen is ignored when matching language codes.

Raises *ValueError* if no matching language is found.

code
Language code like 'fi' or 'pt-BR'.

Got based on the class name. If the class name is two characters (or less), the code is just the name in lower case. If it is longer, a hyphen is added and the remainder of the class name is upper-cased.

This special property can be accessed also directly from the class.

name
Language name like 'Finnish' or 'Brazilian Portuguese'.

Got from the first line of the class docstring.

This special property can be accessed also directly from the class.

headers

settings

bdd_prefixes

class robot.conf.languages.**En**
Bases: *robot.conf.languages.Language*

English

settings_header = 'Settings'

variables_header = 'Variables'

test_cases_header = 'Test Cases'

tasks_header = 'Tasks'

keywords_header = 'Keywords'

comments_header = 'Comments'

library_setting = 'Library'

resource_setting = 'Resource'

variables_setting = 'Variables'

documentation_setting = 'Documentation'

metadata_setting = 'Metadata'

suite_setup_setting = 'Suite Setup'

suite_teardown_setting = 'Suite Teardown'

test_setup_setting = 'Test Setup'

task_setup_setting = 'Task Setup'

test_teardown_setting = 'Test Teardown'

task_teardown_setting = 'Task Teardown'

test_template_setting = 'Test Template'

task_template_setting = 'Task Template'

test_timeout_setting = 'Test Timeout'

task_timeout_setting = 'Task Timeout'

test_tags_setting = 'Test Tags'

task_tags_setting = 'Task Tags'

keyword_tags_setting = 'Keyword Tags'

setup_setting = 'Setup'

teardown_setting = 'Teardown'

template_setting = 'Template'

tags_setting = 'Tags'

timeout_setting = 'Timeout'

arguments_setting = 'Arguments'

given_prefixes = ['Given']

when_prefixes = ['When']

then_prefixes = ['Then']

and_prefixes = ['And']

but_prefixes = ['But']

true_strings = ['True', 'Yes', 'On']

false_strings = ['False', 'No', 'Off']

bdd_prefixes

code = 'en'

classmethod from_name(*name*)
Return language class based on given *name*.

Name can either be a language name (e.g. 'Finnish' or 'Brazilian Portuguese') or a language code (e.g. 'fi' or 'pt-BR'). Matching is case and space insensitive and the hyphen is ignored when matching language codes.

Raises *ValueError* if no matching language is found.

headers

name = 'English'

settings

class robot.conf.languages.**Cs**
Bases: *robot.conf.languages.Language*

Czech

settings_header = 'Nastavení'

variables_header = 'Proměnné'

test_cases_header = 'Testovací případy'

tasks_header = 'Úlohy'

keywords_header = 'Klíčová slova'

comments_header = 'Komentáře'

library_setting = 'Knihovna'

resource_setting = 'Zdroj'

variables_setting = 'Proměnná'

documentation_setting = 'Dokumentace'

metadata_setting = 'Metadata'

suite_setup_setting = 'Příprava sady'

suite_teardown_setting = 'Ukončení sady'

test_setup_setting = 'Příprava testu'

test_teardown_setting = 'Ukončení testu'

test_template_setting = 'Šablona testu'

test_timeout_setting = 'Časový limit testu'

test_tags_setting = 'Štítky testů'

task_setup_setting = 'Příprava úlohy'

task_teardown_setting = 'Ukončení úlohy'

task_template_setting = 'Šablona úlohy'

task_timeout_setting = 'Časový limit úlohy'

task_tags_setting = 'Štítky úloh'

keyword_tags_setting = 'Štítky klíčových slov'

tags_setting = 'Štítky'

setup_setting = 'Příprava'

teardown_setting = 'Ukončení'

template_setting = 'Šablona'

timeout_setting = 'Časový limit'

arguments_setting = 'Argumenty'

given_prefixes = ['Pokud']

when_prefixes = ['Když']

then_prefixes = ['Pak']

and_prefixes = ['A']

but_prefixes = ['Ale']

true_strings = ['Pravda', 'Ano', 'Zapnuto']

false_strings = ['Nepravda', 'Ne', 'Vypnuto', 'Nic']

bdd_prefixes

code = 'cs'

classmethod from_name(*name*)

Return language class based on given *name*.

Name can either be a language name (e.g. 'Finnish' or 'Brazilian Portuguese') or a language code (e.g. 'fi' or 'pt-BR'). Matching is case and space insensitive and the hyphen is ignored when matching language codes.

Raises *ValueError* if no matching language is found.

headers

name = 'Czech'

settings

class robot.conf.languages.**Nl**
Bases: *robot.conf.languages.Language*

Dutch

settings_header = 'Instellingen'

variables_header = 'Variabelen'

test_cases_header = 'Testgevallen'

tasks_header = 'Taken'

keywords_header = 'Sleutelwoorden'

comments_header = 'Opmerkingen'

library_setting = 'Bibliotheek'

resource_setting = 'Resource'

variables_setting = 'Variabele'

documentation_setting = 'Documentatie'

metadata_setting = 'Metadata'

suite_setup_setting = 'Suite Preconditie'

suite_teardown_setting = 'Suite Postconditie'

test_setup_setting = 'Test Preconditie'

test_teardown_setting = 'Test Postconditie'

test_template_setting = 'Test Sjabloon'

test_timeout_setting = 'Test Time-out'

test_tags_setting = 'Test Labels'

task_setup_setting = 'Taak Preconditie'

task_teardown_setting = 'Taak Postconditie'

task_template_setting = 'Taak Sjabloon'

task_timeout_setting = 'Taak Time-out'

task_tags_setting = 'Taak Labels'

keyword_tags_setting = 'Sleutelwoord Labels'

tags_setting = 'Labels'

setup_setting = 'Preconditie'

teardown_setting = 'Postconditie'

template_setting = 'Sjabloon'

timeout_setting = 'Time-out'

arguments_setting = 'Parameters'

given_prefixes = ['Stel', 'Gegeven']

when_prefixes = ['Als']

then_prefixes = ['Dan']

and_prefixes = ['En']

but_prefixes = ['Maar']

true_strings = ['Waar', 'Ja', 'Aan']

false_strings = ['Onwaar', 'Nee', 'Uit', 'Geen']

bdd_prefixes

code = 'nl'

classmethod from_name(*name*)
Return language class based on given *name*.

Name can either be a language name (e.g. 'Finnish' or 'Brazilian Portuguese') or a language code (e.g. 'fi' or 'pt-BR'). Matching is case and space insensitive and the hyphen is ignored when matching language codes.

Raises *ValueError* if no matching language is found.

headers

name = 'Dutch'

settings

class robot.conf.languages.**Bs**
Bases: *robot.conf.languages.Language*

Bosnian

settings_header = 'Postavke'

variables_header = 'Varijable'

test_cases_header = 'Test Cases'

tasks_header = 'Taskovi'

keywords_header = 'Keywords'

comments_header = 'Komentari'

library_setting = 'Biblioteka'

resource_setting = 'Resursi'

variables_setting = 'Varijable'

documentation_setting = 'Dokumentacija'

metadata_setting = 'Metadata'

suite_setup_setting = 'Suite Postavke'

suite_teardown_setting = 'Suite Teardown'

test_setup_setting = 'Test Postavke'

test_teardown_setting = 'Test Teardown'

test_template_setting = 'Test Template'

test_timeout_setting = 'Test Timeout'

test_tags_setting = 'Test Tagovi'

task_setup_setting = 'Task Postavke'

task_teardown_setting = 'Task Teardown'

task_template_setting = 'Task Template'

task_timeout_setting = 'Task Timeout'

task_tags_setting = 'Task Tagovi'

keyword_tags_setting = 'Keyword Tagovi'

tags_setting = 'Tagovi'

setup_setting = 'Postavke'

teardown_setting = 'Teardown'

template_setting = 'Template'

timeout_setting = 'Timeout'

arguments_setting = 'Argumenti'

given_prefixes = ['Uslovno']

when_prefixes = ['Kada']

then_prefixes = ['Tada']

and_prefixes = ['I']

but_prefixes = ['Ali']

bdd_prefixes

code = 'bs'

false_strings = []

classmethod from_name(*name*)
Return language class based on given *name*.

Name can either be a language name (e.g. 'Finnish' or 'Brazilian Portuguese') or a language code (e.g. 'fi' or 'pt-BR'). Matching is case and space insensitive and the hyphen is ignored when matching language codes.

Raises *ValueError* if no matching language is found.

headers

name = 'Bosnian'

settings

true_strings = []

class robot.conf.languages.**Fi**
Bases: *robot.conf.languages.Language*

Finnish

settings_header = 'Asetukset'

variables_header = 'Muuttujat'

```
    test_cases_header = 'Testit'
    tasks_header = 'Tehtävät'
    keywords_header = 'Avainsanat'
    comments_header = 'Kommentit'
    library_setting = 'Kirjasto'
    resource_setting = 'Resurssi'
    variables_setting = 'Muuttujat'
    documentation_setting = 'Dokumentaatio'
    metadata_setting = 'Metatiedot'
    suite_setup_setting = 'Setin Alustus'
    suite_teardown_setting = 'Setin Alasajo'
    test_setup_setting = 'Testin Alustus'
    task_setup_setting = 'Tehtävän Alustus'
    test_teardown_setting = 'Testin Alasajo'
    task_teardown_setting = 'Tehtävän Alasajo'
    test_template_setting = 'Testin Malli'
    task_template_setting = 'Tehtävän Malli'
    test_timeout_setting = 'Testin Aikaraja'
    task_timeout_setting = 'Tehtävän Aikaraja'
    test_tags_setting = 'Testin Tagit'
    task_tags_setting = 'Tehtävän Tagit'
    keyword_tags_setting = 'Avainsanan Tagit'
    tags_setting = 'Tagit'
    setup_setting = 'Alustus'
    teardown_setting = 'Alasajo'
    template_setting = 'Malli'
    timeout_setting = 'Aikaraja'
    arguments_setting = 'Argumentit'
    given_prefixes = ['Oletetaan']
    when_prefixes = ['Kun']
    then_prefixes = ['Niin']
    and_prefixes = ['Ja']
    but_prefixes = ['Mutta']
    true_strings = ['Tosi', 'Kyllä', 'Päällä']
    false_strings = ['Epätosi', 'Ei', 'Pois']
    bdd_prefixes
```

code = 'fi'

classmethod from_name(*name*)
Return language class based on given *name*.

Name can either be a language name (e.g. 'Finnish' or 'Brazilian Portuguese') or a language code (e.g. 'fi' or 'pt-BR'). Matching is case and space insensitive and the hyphen is ignored when matching language codes.

Raises *ValueError* if no matching language is found.

headers

name = 'Finnish'

settings

class robot.conf.languages.**Fr**
Bases: *robot.conf.languages.Language*

French

settings_header = 'Paramètres'

variables_header = 'Variables'

test_cases_header = 'Unités de test'

tasks_header = 'Tâches'

keywords_header = 'Mots-clés'

comments_header = 'Commentaires'

library_setting = 'Bibliothèque'

resource_setting = 'Ressource'

variables_setting = 'Variable'

documentation_setting = 'Documentation'

metadata_setting = 'Méta-donnée'

suite_setup_setting = 'Mise en place de suite'

suite_teardown_setting = 'Démontage de suite'

test_setup_setting = 'Mise en place de test'

test_teardown_setting = 'Démontage de test'

test_template_setting = 'Modèle de test'

test_timeout_setting = 'Délai de test'

test_tags_setting = 'Étiquette de test'

task_setup_setting = 'Mise en place de tâche'

task_teardown_setting = 'Démontage de test'

task_template_setting = 'Modèle de tâche'

task_timeout_setting = 'Délai de tâche'

task_tags_setting = 'Étiquette de tâche'

keyword_tags_setting = 'Etiquette de mot-clé'

tags_setting = 'Étiquette'

setup_setting = 'Mise en place'

teardown_setting = 'Démontage'

template_setting = 'Modèle'

timeout_setting = "Délai d'attente"

arguments_setting = 'Arguments'

given_prefixes = ['Étant donné']

when_prefixes = ['Lorsque']

then_prefixes = ['Alors']

and_prefixes = ['Et']

but_prefixes = ['Mais']

true_strings = ['Vrai', 'Oui', 'Actif']

false_strings = ['Faux', 'Non', 'Désactivé', 'Aucun']

bdd_prefixes

code = 'fr'

classmethod from_name(*name*)
Return language class based on given *name*.

Name can either be a language name (e.g. 'Finnish' or 'Brazilian Portuguese') or a language code (e.g. 'fi' or 'pt-BR'). Matching is case and space insensitive and the hyphen is ignored when matching language codes.

Raises *ValueError* if no matching language is found.

headers

name = 'French'

settings

class robot.conf.languages.**De**
Bases: *robot.conf.languages.Language*

German

settings_header = 'Einstellungen'

variables_header = 'Variablen'

test_cases_header = 'Testfälle'

tasks_header = 'Aufgaben'

keywords_header = 'Schlüsselwörter'

comments_header = 'Kommentare'

library_setting = 'Bibliothek'

resource_setting = 'Ressource'

variables_setting = 'Variablen'

documentation_setting = 'Dokumentation'

metadata_setting = 'Metadaten'

suite_setup_setting = 'Suitevorbereitung'

suite_teardown_setting = 'Suitenachbereitung'

test_setup_setting = 'Testvorbereitung'

test_teardown_setting = 'Testnachbereitung'

test_template_setting = 'Testvorlage'

test_timeout_setting = 'Testzeitlimit'

test_tags_setting = 'Testmarker'

task_setup_setting = 'Aufgabenvorbereitung'

task_teardown_setting = 'Aufgabennachbereitung'

task_template_setting = 'Aufgabenvorlage'

task_timeout_setting = 'Aufgabenzeitlimit'

task_tags_setting = 'Aufgabenmarker'

keyword_tags_setting = 'Schlüsselwortmarker'

tags_setting = 'Marker'

setup_setting = 'Vorbereitung'

teardown_setting = 'Nachbereitung'

template_setting = 'Vorlage'

timeout_setting = 'Zeitlimit'

arguments_setting = 'Argumente'

given_prefixes = ['Angenommen']

when_prefixes = ['Wenn']

then_prefixes = ['Dann']

and_prefixes = ['Und']

but_prefixes = ['Aber']

true_strings = ['Wahr', 'Ja', 'An', 'Ein']

false_strings = ['Falsch', 'Nein', 'Aus', 'Unwahr']

bdd_prefixes

code = 'de'

classmethod from_name(*name*)

Return language class based on given *name*.

Name can either be a language name (e.g. 'Finnish' or 'Brazilian Portuguese') or a language code (e.g. 'fi' or 'pt-BR'). Matching is case and space insensitive and the hyphen is ignored when matching language codes.

Raises *ValueError* if no matching language is found.

headers

name = 'German'

settings

class robot.conf.languages.**PtBr**

Bases: *robot.conf.languages.Language*

Brazilian Portuguese

settings_header = 'Configurações'

variables_header = 'Variáveis'

test_cases_header = 'Casos de Teste'

tasks_header = 'Tarefas'

keywords_header = 'Palavras-Chave'

comments_header = 'Comentários'

library_setting = 'Biblioteca'

resource_setting = 'Recurso'

variables_setting = 'Variável'

documentation_setting = 'Documentação'

metadata_setting = 'Metadados'

suite_setup_setting = 'Configuração da Suíte'

suite_teardown_setting = 'Finalização de Suíte'

test_setup_setting = 'Inicialização de Teste'

test_teardown_setting = 'Finalização de Teste'

test_template_setting = 'Modelo de Teste'

test_timeout_setting = 'Tempo Limite de Teste'

test_tags_setting = 'Test Tags'

task_setup_setting = 'Inicialização de Tarefa'

task_teardown_setting = 'Finalização de Tarefa'

task_template_setting = 'Modelo de Tarefa'

task_timeout_setting = 'Tempo Limite de Tarefa'

task_tags_setting = 'Task Tags'

keyword_tags_setting = 'Keyword Tags'

tags_setting = 'Etiquetas'

setup_setting = 'Inicialização'

teardown_setting = 'Finalização'

template_setting = 'Modelo'

timeout_setting = 'Tempo Limite'

arguments_setting = 'Argumentos'

given_prefixes = ['Dado']

when_prefixes = ['Quando']

then_prefixes = ['Então']

and_prefixes = ['E']

but_prefixes = ['Mas']

true_strings = ['Verdadeiro', 'Verdade', 'Sim', 'Ligado']

false_strings = ['Falso', 'Não', 'Desligado', 'Desativado', 'Nada']

bdd_prefixes

code = 'pt-BR'

classmethod from_name(*name*)
Return language class based on given *name*.

Name can either be a language name (e.g. 'Finnish' or 'Brazilian Portuguese') or a language code (e.g. 'fi' or 'pt-BR'). Matching is case and space insensitive and the hyphen is ignored when matching language codes.

Raises *ValueError* if no matching language is found.

headers

name = 'Brazilian Portuguese'

settings

class robot.conf.languages.**Pt**
Bases: *robot.conf.languages.Language*

Portuguese

settings_header = 'Definições'

variables_header = 'Variáveis'

test_cases_header = 'Casos de Teste'

tasks_header = 'Tarefas'

keywords_header = 'Palavras-Chave'

comments_header = 'Comentários'

library_setting = 'Biblioteca'

resource_setting = 'Recurso'

variables_setting = 'Variável'

documentation_setting = 'Documentação'

metadata_setting = 'Metadados'

suite_setup_setting = 'Inicialização de Suíte'

suite_teardown_setting = 'Finalização de Suíte'

test_setup_setting = 'Inicialização de Teste'

test_teardown_setting = 'Finalização de Teste'

test_template_setting = 'Modelo de Teste'

test_timeout_setting = 'Tempo Limite de Teste'

test_tags_setting = 'Etiquetas de Testes'

task_setup_setting = 'Inicialização de Tarefa'

task_teardown_setting = 'Finalização de Tarefa'

task_template_setting = 'Modelo de Tarefa'

task_timeout_setting = 'Tempo Limite de Tarefa'

task_tags_setting = 'Etiquetas de Tarefas'

keyword_tags_setting = 'Etiquetas de Palavras-Chave'

tags_setting = 'Etiquetas'

setup_setting = 'Inicialização'

teardown_setting = 'Finalização'

template_setting = 'Modelo'

timeout_setting = 'Tempo Limite'

arguments_setting = 'Argumentos'

given_prefixes = ['Dado']

when_prefixes = ['Quando']

then_prefixes = ['Então']

and_prefixes = ['E']

but_prefixes = ['Mas']

true_strings = ['Verdadeiro', 'Verdade', 'Sim', 'Ligado']

false_strings = ['Falso', 'Não', 'Desligado', 'Desativado', 'Nada']

bdd_prefixes

code = 'pt'

classmethod from_name(*name*)

Return language class based on given *name*.

Name can either be a language name (e.g. 'Finnish' or 'Brazilian Portuguese') or a language code (e.g. 'fi' or 'pt-BR'). Matching is case and space insensitive and the hyphen is ignored when matching language codes.

Raises *ValueError* if no matching language is found.

headers

name = 'Portuguese'

settings

class robot.conf.languages.**Th**

Bases: *robot.conf.languages.Language*

Thai

settings_header = ''

variables_header = ''

test_cases_header = ''

tasks_header = ''

```
keywords_header = ''
comments_header = ''
library_setting = ''
resource_setting = ''
variables_setting = ''
documentation_setting = ''
metadata_setting = ''
suite_setup_setting = ''
suite_teardown_setting = ''
test_setup_setting = ''
task_setup_setting = ''
test_teardown_setting = ''
task_teardown_setting = ''
test_template_setting = ''
task_template_setting = ''
test_timeout_setting = ''
task_timeout_setting = ''
test_tags_setting = ''
task_tags_setting = ''
keyword_tags_setting = ''
setup_setting = ''
teardown_setting = ''
template_setting = ''
tags_setting = ''
timeout_setting = ''
arguments_setting = ''
given_prefixes = ['']
when_prefixes = ['']
then_prefixes = ['']
and_prefixes = ['']
but_prefixes = ['']
bdd_prefixes
code = 'th'
false_strings = []
```

classmethod from_name(*name*)
Return language class based on given *name*.

Name can either be a language name (e.g. 'Finnish' or 'Brazilian Portuguese') or a language code (e.g. 'fi' or 'pt-BR'). Matching is case and space insensitive and the hyphen is ignored when matching language codes.

Raises *ValueError* if no matching language is found.

headers

name = 'Thai'

settings

true_strings = []

class robot.conf.languages.**Pl**
Bases: *robot.conf.languages.Language*

Polish

settings_header = 'Ustawienia'

variables_header = 'Zmienne'

test_cases_header = 'Przypadki testowe'

tasks_header = 'Zadania'

keywords_header = 'Słowa kluczowe'

comments_header = 'Komentarze'

library_setting = 'Biblioteka'

resource_setting = 'Zasób'

variables_setting = 'Zmienne'

documentation_setting = 'Dokumentacja'

metadata_setting = 'Metadane'

suite_setup_setting = 'Inicjalizacja zestawu'

suite_teardown_setting = 'Ukończenie zestawu'

test_setup_setting = 'Inicjalizacja testu'

test_teardown_setting = 'Ukończenie testu'

test_template_setting = 'Szablon testu'

test_timeout_setting = 'Limit czasowy testu'

test_tags_setting = 'Znaczniki testu'

task_setup_setting = 'Inicjalizacja zadania'

task_teardown_setting = 'Ukończenie zadania'

task_template_setting = 'Szablon zadania'

task_timeout_setting = 'Limit czasowy zadania'

task_tags_setting = 'Znaczniki zadania'

keyword_tags_setting = 'Znaczniki słowa kluczowego'

tags_setting = 'Znaczniki'

setup_setting = 'Inicjalizacja'

teardown_setting = 'Ukończenie'

template_setting = 'Szablon'

timeout_setting = 'Limit czasowy'

arguments_setting = 'Argumenty'

given_prefixes = ['Zakładając', 'Zakładając, że', 'Mając']

when_prefixes = ['Jeżeli', 'Jeśli', 'Gdy', 'Kiedy']

then_prefixes = ['Wtedy']

and_prefixes = ['Oraz', 'I']

but_prefixes = ['Ale']

true_strings = ['Prawda', 'Tak', 'Włączone']

false_strings = ['Fałsz', 'Nie', 'Wyłączone', 'Nic']

bdd_prefixes

code = 'pl'

classmethod from_name(*name*)

Return language class based on given *name*.

Name can either be a language name (e.g. 'Finnish' or 'Brazilian Portuguese') or a language code (e.g. 'fi' or 'pt-BR'). Matching is case and space insensitive and the hyphen is ignored when matching language codes.

Raises *ValueError* if no matching language is found.

headers

name = 'Polish'

settings

class robot.conf.languages.**Uk**

Bases: *robot.conf.languages.Language*

Ukrainian

settings_header = ''

variables_header = ''

test_cases_header = '-'

tasks_header = ''

keywords_header = ' '

comments_header = ''

library_setting = ''

resource_setting = ''

variables_setting = ''

documentation_setting = ''

metadata_setting = ''

suite_setup_setting = ' Suite'

suite_teardown_setting = ' Suite'

test_setup_setting = ' '

test_teardown_setting = ' y'

test_template_setting = ' '

test_timeout_setting = ' '

test_tags_setting = ' '

task_setup_setting = ' '

task_teardown_setting = ' '

task_template_setting = ' '

task_timeout_setting = ' '

task_tags_setting = ' '

keyword_tags_setting = ' '

tags_setting = ''

setup_setting = ''

teardown_setting = 'C   '

template_setting = ''

timeout_setting = ' '

arguments_setting = ''

given_prefixes = ['']

when_prefixes = ['']

then_prefixes = ['']

and_prefixes = ['']

but_prefixes = ['']

bdd_prefixes

code = 'uk'

false_strings = []

classmethod from_name(*name*)
Return language class based on given *name*.

Name can either be a language name (e.g. 'Finnish' or 'Brazilian Portuguese') or a language code (e.g. 'fi' or 'pt-BR'). Matching is case and space insensitive and the hyphen is ignored when matching language codes.

Raises *ValueError* if no matching language is found.

headers

name = 'Ukrainian'

settings

```
    true_strings = []
```

class robot.conf.languages.**Es**

Bases: *robot.conf.languages.Language*

Spanish

```
    settings_header = 'Configuraciones'
    variables_header = 'Variables'
    test_cases_header = 'Casos de prueba'
    tasks_header = 'Tareas'
    keywords_header = 'Palabras clave'
    comments_header = 'Comentarios'
    library_setting = 'Biblioteca'
    resource_setting = 'Recursos'
    variables_setting = 'Variable'
    documentation_setting = 'Documentación'
    metadata_setting = 'Metadatos'
    suite_setup_setting = 'Configuración de la Suite'
    suite_teardown_setting = 'Desmontaje de la Suite'
    test_setup_setting = 'Configuración de prueba'
    test_teardown_setting = 'Desmontaje de la prueba'
    test_template_setting = 'Plantilla de prueba'
    test_timeout_setting = 'Tiempo de espera de la prueba'
    test_tags_setting = 'Etiquetas de la prueba'
    task_setup_setting = 'Configuración de tarea'
    task_teardown_setting = 'Desmontaje de tareas'
    task_template_setting = 'Plantilla de tareas'
    task_timeout_setting = 'Tiempo de espera de las tareas'
    task_tags_setting = 'Etiquetas de las tareas'
    keyword_tags_setting = 'Etiquetas de palabras clave'
    tags_setting = 'Etiquetas'
    setup_setting = 'Configuración'
    teardown_setting = 'Desmontaje'
    template_setting = 'Plantilla'
    timeout_setting = 'Tiempo agotado'
    arguments_setting = 'Argumentos'
    given_prefixes = ['Dado']
    when_prefixes = ['Cuando']
```

then_prefixes = ['Entonces']

and_prefixes = ['Y']

but_prefixes = ['Pero']

true_strings = ['Verdadero', 'Si', 'On']

false_strings = ['Falso', 'No', 'Off', 'Ninguno']

bdd_prefixes

code = 'es'

classmethod from_name(*name*)

Return language class based on given *name*.

Name can either be a language name (e.g. 'Finnish' or 'Brazilian Portuguese') or a language code (e.g. 'fi' or 'pt-BR'). Matching is case and space insensitive and the hyphen is ignored when matching language codes.

Raises *ValueError* if no matching language is found.

headers

name = 'Spanish'

settings

class robot.conf.languages.**Ru**

Bases: *robot.conf.languages.Language*

Russian

settings_header = ''

variables_header = ''

test_cases_header = ' '

tasks_header = ''

keywords_header = ' '

comments_header = ''

library_setting = ''

resource_setting = ''

variables_setting = ''

documentation_setting = ''

metadata_setting = ''

suite_setup_setting = ' '

suite_teardown_setting = ' '

test_setup_setting = ' '

test_teardown_setting = ' '

test_template_setting = ' '

test_timeout_setting = ' '

test_tags_setting = ' '

task_setup_setting = ' '

task_teardown_setting = ' '

task_template_setting = ' '

task_timeout_setting = ' '

task_tags_setting = ' '

keyword_tags_setting = ' '

tags_setting = ''

setup_setting = ''

teardown_setting = ''

template_setting = ''

timeout_setting = ''

arguments_setting = ''

given_prefixes = ['']

when_prefixes = ['']

then_prefixes = ['']

and_prefixes = ['']

but_prefixes = ['']

bdd_prefixes

code = 'ru'

false_strings = []

classmethod from_name(*name*)
Return language class based on given *name*.

Name can either be a language name (e.g. 'Finnish' or 'Brazilian Portuguese') or a language code (e.g. 'fi' or 'pt-BR'). Matching is case and space insensitive and the hyphen is ignored when matching language codes.

Raises *ValueError* if no matching language is found.

headers

name = 'Russian'

settings

true_strings = []

class robot.conf.languages.**ZhCn**
Bases: *robot.conf.languages.Language*

Chinese Simplified

settings_header = ''

variables_header = ''

test_cases_header = ''

tasks_header = ''

```
keywords_header = ''
comments_header = ''
library_setting = ''
resource_setting = ''
variables_setting = ''
documentation_setting = ''
metadata_setting = ''
suite_setup_setting = ''
suite_teardown_setting = ''
test_setup_setting = ''
test_teardown_setting = ''
test_template_setting = ''
test_timeout_setting = ''
test_tags_setting = ''
task_setup_setting = ''
task_teardown_setting = ''
task_template_setting = ''
task_timeout_setting = ''
task_tags_setting = ''
keyword_tags_setting = ''
tags_setting = ''
setup_setting = ''
teardown_setting = ''
template_setting = ''
timeout_setting = ''
arguments_setting = ''
given_prefixes = ['']
when_prefixes = ['']
then_prefixes = ['']
and_prefixes = ['']
but_prefixes = ['']
true_strings = ['', '', '']
false_strings = ['', '', '', '']
bdd_prefixes
code = 'zh-CN'
```

classmethod from_name(*name*)
Return language class based on given *name*.

Name can either be a language name (e.g. 'Finnish' or 'Brazilian Portuguese') or a language code (e.g. 'fi' or 'pt-BR'). Matching is case and space insensitive and the hyphen is ignored when matching language codes.

Raises *ValueError* if no matching language is found.

headers

name = 'Chinese Simplified'

settings

class robot.conf.languages.**ZhTw**
Bases: *robot.conf.languages.Language*

Chinese Traditional

settings_header = ''

variables_header = ''

test_cases_header = ''

tasks_header = ''

keywords_header = ''

comments_header = ''

library_setting = ''

resource_setting = ''

variables_setting = ''

documentation_setting = ''

metadata_setting = ''

suite_setup_setting = ''

suite_teardown_setting = ''

test_setup_setting = ''

test_teardown_setting = ''

test_template_setting = ''

test_timeout_setting = ''

test_tags_setting = ''

task_setup_setting = ''

task_teardown_setting = ''

task_template_setting = ''

task_timeout_setting = ''

task_tags_setting = ''

keyword_tags_setting = ''

tags_setting = ''

setup_setting = ''

teardown_setting = ''

template_setting = ''

timeout_setting = ''

arguments_setting = ''

given_prefixes = ['']

when_prefixes = ['']

then_prefixes = ['']

and_prefixes = ['']

but_prefixes = ['']

true_strings = ['', '', '']

false_strings = ['', '', '', '']

bdd_prefixes

code = 'zh-TW'

classmethod from_name(*name*)
Return language class based on given *name*.

Name can either be a language name (e.g. 'Finnish' or 'Brazilian Portuguese') or a language code (e.g. 'fi' or 'pt-BR'). Matching is case and space insensitive and the hyphen is ignored when matching language codes.

Raises *ValueError* if no matching language is found.

headers

name = 'Chinese Traditional'

settings

class robot.conf.languages.**Tr**
Bases: *robot.conf.languages.Language*

Turkish

settings_header = 'Ayarlar'

variables_header = 'Değişkenler'

test_cases_header = 'Test Durumları'

tasks_header = 'Görevler'

keywords_header = 'Anahtar Kelimeler'

comments_header = 'Yorumlar'

library_setting = 'Kütüphane'

resource_setting = 'Kaynak'

variables_setting = 'Değişkenler'

documentation_setting = 'Dokümantasyon'

metadata_setting = 'Üstveri'

suite_setup_setting = 'Takım Kurulumu'

suite_teardown_setting = 'Takım Bitişi'

test_setup_setting = 'Test Kurulumu'

task_setup_setting = 'Görev Kurulumu'

test_teardown_setting = 'Test Bitişi'

task_teardown_setting = 'Görev Bitişi'

test_template_setting = 'Test Taslağı'

task_template_setting = 'Görev Taslağı'

test_timeout_setting = 'Test Zaman Aşımı'

task_timeout_setting = 'Görev Zaman Aşımı'

test_tags_setting = 'Test Etiketleri'

task_tags_setting = 'Görev Etiketleri'

keyword_tags_setting = 'Anahtar Kelime Etiketleri'

setup_setting = 'Kurulum'

teardown_setting = 'Bitiş'

template_setting = 'Taslak'

tags_setting = 'Etiketler'

timeout_setting = 'Zaman Aşımı'

arguments_setting = 'Argümanlar'

given_prefixes = ['Diyelim ki']

when_prefixes = ['Eğer ki']

then_prefixes = ['O zaman']

and_prefixes = ['Ve']

but_prefixes = ['Ancak']

true_strings = ['Doğru', 'Evet', 'Açik']

false_strings = ['Yanliş', 'Hayir', 'Kapali']

bdd_prefixes

code = 'tr'

classmethod from_name(*name*)

Return language class based on given *name*.

Name can either be a language name (e.g. 'Finnish' or 'Brazilian Portuguese') or a language code (e.g. 'fi' or 'pt-BR'). Matching is case and space insensitive and the hyphen is ignored when matching language codes.

Raises *ValueError* if no matching language is found.

headers

name = 'Turkish'

settings

class robot.conf.languages.**Sv**

Bases: *robot.conf.languages.Language*

Swedish

settings_header = 'Inställningar'

variables_header = 'Variabler'

test_cases_header = 'Testfall'

tasks_header = 'Taskar'

keywords_header = 'Nyckelord'

comments_header = 'Kommentarer'

library_setting = 'Bibliotek'

resource_setting = 'Resurs'

variables_setting = 'Variabel'

documentation_setting = 'Dokumentation'

metadata_setting = 'Metadata'

suite_setup_setting = 'Svit konfigurering'

suite_teardown_setting = 'Svit nedrivning'

test_setup_setting = 'Test konfigurering'

test_teardown_setting = 'Test nedrivning'

test_template_setting = 'Test mall'

test_timeout_setting = 'Test timeout'

test_tags_setting = 'Test taggar'

task_setup_setting = 'Task konfigurering'

task_teardown_setting = 'Task nedrivning'

task_template_setting = 'Task mall'

task_timeout_setting = 'Task timeout'

task_tags_setting = 'Arbetsuppgift taggar'

keyword_tags_setting = 'Nyckelord taggar'

tags_setting = 'Taggar'

setup_setting = 'Konfigurering'

teardown_setting = 'Nedrivning'

template_setting = 'Mall'

timeout_setting = 'Timeout'

arguments_setting = 'Argument'

given_prefixes = ['Givet']

when_prefixes = ['När']

then_prefixes = ['Då']

and_prefixes = ['Och']

but_prefixes = ['Men']

true_strings = ['Sant', 'Ja', 'På']

false_strings = ['Falskt', 'Nej', 'Av', 'Ingen']

bdd_prefixes

code = 'sv'

classmethod from_name(*name*)
Return language class based on given *name*.

Name can either be a language name (e.g. 'Finnish' or 'Brazilian Portuguese') or a language code (e.g. 'fi' or 'pt-BR'). Matching is case and space insensitive and the hyphen is ignored when matching language codes.

Raises *ValueError* if no matching language is found.

headers

name = 'Swedish'

settings

class robot.conf.languages.**Bg**
Bases: *robot.conf.languages.Language*

Bulgarian

settings_header = ''

variables_header = ''

test_cases_header = ' '

tasks_header = ''

keywords_header = ' '

comments_header = ''

library_setting = ''

resource_setting = ''

variables_setting = ''

documentation_setting = ''

metadata_setting = ''

suite_setup_setting = ' '

suite_teardown_setting = ' '

test_setup_setting = ' '

test_teardown_setting = ' '

test_template_setting = ' '

test_timeout_setting = ' '

test_tags_setting = ' '

task_setup_setting = ' '

task_teardown_setting = ' '

task_template_setting = ' '

task_timeout_setting = ' '

task_tags_setting = ' '

keyword_tags_setting = '   '

tags_setting = ''

setup_setting = ' '

teardown_setting = ''

template_setting = ''

timeout_setting = ''

arguments_setting = ''

given_prefixes = [' ']

when_prefixes = ['']

then_prefixes = ['']

and_prefixes = ['']

but_prefixes = ['']

true_strings = ['', '', '']

false_strings = ['', '', '', '']

bdd_prefixes

code = 'bg'

classmethod from_name(*name*)

Return language class based on given *name*.

Name can either be a language name (e.g. 'Finnish' or 'Brazilian Portuguese') or a language code (e.g. 'fi' or 'pt-BR'). Matching is case and space insensitive and the hyphen is ignored when matching language codes.

Raises *ValueError* if no matching language is found.

headers

name = 'Bulgarian'

settings

class robot.conf.languages.**Ro**

Bases: *robot.conf.languages.Language*

Romanian

settings_header = 'Setari'

variables_header = 'Variabile'

test_cases_header = 'Cazuri De Test'

tasks_header = 'Sarcini'

keywords_header = 'Cuvinte Cheie'

```
comments_header = 'Comentarii'
library_setting = 'Librarie'
resource_setting = 'Resursa'
variables_setting = 'Variabila'
documentation_setting = 'Documentatie'
metadata_setting = 'Metadate'
suite_setup_setting = 'Configurare De Suita'
suite_teardown_setting = 'Configurare De Intrerupere'
test_setup_setting = 'Setare De Test'
test_teardown_setting = 'Inrerupere De Test'
test_template_setting = 'Sablon De Test'
test_timeout_setting = 'Timp Expirare Test'
test_tags_setting = 'Taguri De Test'
task_setup_setting = 'Configuarare activitate'
task_teardown_setting = 'Intrerupere activitate'
task_template_setting = 'Sablon de activitate'
task_timeout_setting = 'Timp de expirare activitate'
task_tags_setting = 'Etichete activitate'
keyword_tags_setting = 'Etichete metode'
tags_setting = 'Etichete'
setup_setting = 'Setare'
teardown_setting = 'Intrerupere'
template_setting = 'Sablon'
timeout_setting = 'Expirare'
arguments_setting = 'Argumente'
given_prefixes = ['Fie ca']
when_prefixes = ['Cand']
then_prefixes = ['Atunci']
and_prefixes = ['Si']
but_prefixes = ['Dar']
true_strings = ['Adevarat', 'Da', 'Cand']
false_strings = ['Fals', 'Nu', 'Oprit', 'Niciun']
bdd_prefixes
code = 'ro'
```

classmethod from_name(*name*)
Return language class based on given *name*.

Name can either be a language name (e.g. 'Finnish' or 'Brazilian Portuguese') or a language code (e.g. 'fi' or 'pt-BR'). Matching is case and space insensitive and the hyphen is ignored when matching language codes.

Raises *ValueError* if no matching language is found.

headers

name = 'Romanian'

settings

class robot.conf.languages.**It**
Bases: *robot.conf.languages.Language*

Italian

settings_header = 'Impostazioni'

variables_header = 'Variabili'

test_cases_header = 'Casi Di Test'

tasks_header = 'Attività'

keywords_header = 'Parole Chiave'

comments_header = 'Commenti'

library_setting = 'Libreria'

resource_setting = 'Risorsa'

variables_setting = 'Variabile'

documentation_setting = 'Documentazione'

metadata_setting = 'Metadati'

suite_setup_setting = 'Configurazione Suite'

suite_teardown_setting = 'Distruzione Suite'

test_setup_setting = 'Configurazione Test'

test_teardown_setting = 'Distruzione Test'

test_template_setting = 'Modello Test'

test_timeout_setting = 'Timeout Test'

test_tags_setting = 'Tag Del Test'

task_setup_setting = 'Configurazione Attività'

task_teardown_setting = 'Distruzione Attività'

task_template_setting = 'Modello Attività'

task_timeout_setting = 'Timeout Attività'

task_tags_setting = 'Tag Attività'

keyword_tags_setting = 'Tag Parola Chiave'

tags_setting = 'Tag'

setup_setting = 'Configurazione'

teardown_setting = 'Distruzione'

template_setting = 'Template'

timeout_setting = 'Timeout'

arguments_setting = 'Parametri'

given_prefixes = ['Dato']

when_prefixes = ['Quando']

then_prefixes = ['Allora']

and_prefixes = ['E']

but_prefixes = ['Ma']

true_strings = ['Vero', 'Sì', 'On']

false_strings = ['Falso', 'No', 'Off', 'Nessuno']

bdd_prefixes

code = 'it'

classmethod from_name(*name*)

Return language class based on given *name*.

Name can either be a language name (e.g. 'Finnish' or 'Brazilian Portuguese') or a language code (e.g. 'fi' or 'pt-BR'). Matching is case and space insensitive and the hyphen is ignored when matching language codes.

Raises *ValueError* if no matching language is found.

headers

name = 'Italian'

settings

class robot.conf.languages.**Hi**

Bases: *robot.conf.languages.Language*

Hindi

settings_header = ''

variables_header = ''

test_cases_header = ' '

tasks_header = ' '

keywords_header = ''

comments_header = ''

library_setting = ' '

resource_setting = ''

variables_setting = ''

documentation_setting = ''

metadata_setting = '-'

suite_setup_setting = ' '

suite_teardown_setting = ' '

test_setup_setting = ' '

test_teardown_setting = ' '

test_template_setting = ' '

test_timeout_setting = '   '

test_tags_setting = ' '

task_setup_setting = ' '

task_teardown_setting = ' '

task_template_setting = ' '

task_timeout_setting = ' '

task_tags_setting = ' '

keyword_tags_setting = ' '

tags_setting = ''

setup_setting = ''

teardown_setting = ''

template_setting = ''

timeout_setting = ' '

arguments_setting = ''

given_prefixes = [' ']

when_prefixes = ['']

then_prefixes = ['']

and_prefixes = ['']

but_prefixes = ['']

true_strings = ['', '', '', '']

false_strings = ['', '', '', '', '', '']

bdd_prefixes

code = 'hi'

classmethod from_name(*name*)
Return language class based on given *name*.

Name can either be a language name (e.g. 'Finnish' or 'Brazilian Portuguese') or a language code (e.g. 'fi' or 'pt-BR'). Matching is case and space insensitive and the hyphen is ignored when matching language codes.

Raises *ValueError* if no matching language is found.

headers

name = 'Hindi'

settings

robot.conf.settings module

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

get_rebot_settings()

listeners

debug_file

languages

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 public Libdoc API is exposed via the *robot.libdoc* module.

Submodules

robot.libdocpkg.builder module

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

Generate keyword documentation for the given library, resource or suite file.

Parameters

- **library_or_resource** – Name or path of the library, or path of a resource or a suite file.
- **name** – Set name with the given value.
- **version** – Set version to the given value.
- **doc_format** – Set documentation format to the given value.

Returns *LibraryDoc* instance.

This factory method is the recommended API to generate keyword documentation programmatically. It should be imported via the *robot.libdoc* module.

Example:

```
from robot.libdoc import LibraryDocumentation

lib = LibraryDocumentation('OperatingSystem')
print(lib.name, lib.version)
for kw in lib.keywords:
    print(kw.name)
```

class robot.libdocpkg.builder.**DocumentationBuilder**(*library_or_resource=None*)

Bases: object

Keyword documentation builder.

This is not part of Libdoc's public API. Use *LibraryDocumentation()* instead.

library_or_resource is accepted for backwards compatibility reasons.

It is not used for anything internally and passing it to the builder is considered deprecated starting from RF 6.0.1.

build(*source*)

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_hint*, *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 = {'&': '&', ''': "'", '>': '>', '<': '<', '"': '"

get_shortdoc_from_html(*doc*)

html_to_plain_text(*doc*)

robot.libdocpkg.htmlwriter module

class robot.libdocpkg.htmlwriter.**LibdocHtmlWriter**(*theme=None*)
Bases: object

write(*libdoc*, *output*)

class robot.libdocpkg.htmlwriter.**LibdocModelWriter**(*output*, *libdoc*, *theme=None*)
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

Documentation for a library, a resource file or a suite file.

doc

doc_format

inits
Initializer docs as *KeywordDoc* instances.

keywords
Keyword docs as *KeywordDoc* instances.

type_docs

all_tags

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

convert_docs_to_html()

to_dictionary(*include_private=False*, *theme=None*)

to_json(*indent=None*, *include_private=True*, *theme=None*)

class robot.libdocpkg.model.**KeywordDoc**(*name=''*, *args=None*, *doc=''*, *shortdoc=''*, *tags=()*, *private=False*, *deprecated=False*, *source=None*, *lineno=-1*, *parent=None*)

Bases: robot.utils.sortable.Sortable

Documentation for a single keyword or an initializer.

shortdoc

to_dictionary()

robot.libdocpkg.output module

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

Bases: object

robot.libdocpkg.output.**get_generation_time**()

Return a timestamp that honors *SOURCE_DATE_EPOCH*.

This timestamp is to be used for embedding in output files, so that builds can be made reproducible.

robot.libdocpkg.robotbuilder module

class robot.libdocpkg.robotbuilder.**LibraryDocBuilder**

Bases: object

build(*library*)

class robot.libdocpkg.robotbuilder.**ResourceDocBuilder**

Bases: object

type = 'RESOURCE'

build(*path*)

class robot.libdocpkg.robotbuilder.**SuiteDocBuilder**

Bases: *robot.libdocpkg.robotbuilder.ResourceDocBuilder*

type = 'SUITE'

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*, *theme=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*, *dry_run=False*)

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|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 mode details.

== Variables in expressions ==

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

Actual variables values are also available in the evaluation namespace. They can be accessed using special variable syntax without the curly braces like `$variable`. These variables should never be quoted.

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

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

= Boolean arguments =

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

True examples:

False examples:

= 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] 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] that are more powerful but also more complicated that glob patterns. The regular expression support is implemented using Python's [http://docs.python.org/library/re.html|re module] and its documentation should be consulted for more information about the syntax.

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

= Multiline string comparison =

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

Results in the following error message:

= String representations =

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

- It is not possible to see difference between different objects that have 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 `a` (`\u0061`) and the Cyrillic `а` (`\u0430`). Error messages like `a != а` are not very helpful.
- Some Unicode characters can be represented using [https://en.wikipedia.org/wiki/Unicode_equivalence|different forms]. For example, `ä` can be represented either as a single code point `\u00e4` or using two 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] 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 = '6.0.2'

call_method(*object*, *method_name*, **args*, ***kwargs*)
Calls the named method of the given object with the provided arguments.

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

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#truth|truth value] using Python's `bool()` method.

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

Valid input types are listed below:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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] 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 is more than one keyword 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 `<img src="image.png">` creates an image when `html` is true, but otherwise the message is that exact string. An alternative to using the `html` argument is using the HTML pseudo log level. It logs the message as HTML using the INFO level.

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

The `formatter` argument controls how to format the string representation of the message. Possible values are `str` (default), `repr`, `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|format specification] 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 that is used matches multiple keyword implementations. 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), execution 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 keyword 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.
- Calling this keyword without arguments removes possible search order.
- 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 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-faql 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*, *flags=None*)
Fails if `string` does not match `pattern` as a regular expression.

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

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

Possible flags altering how the expression is parsed (e.g. `re.IGNORECASE`, `re.MULTILINE`) can be given using the `flags` argument (e.g. `flags=IGNORECASE | MULTILINE`) or embedded to the pattern (e.g. `(?im)pattern`).

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.

The `flags` argument is new in Robot Framework 6.0.

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-faql 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_regexp(*string*, *pattern*, *msg=None*, *values=True*, *flags=None*)
Fails if `string` matches `pattern` as a regular expression.

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

should_not_start_with(*str1*, *str2*, *msg=None*, *values=True*, *ignore_case=False*, *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. Providing a value without specifying minutes or seconds, defaults to seconds. 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=0*, *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.

Parameters

- **library** – Name of the library the keyword belongs to.
- **keyword** – Name of the keyword itself.
- **args_to_process** – How many arguments to process normally before passing them to the keyword. Other arguments are not touched at all.
- **deprecation_warning** – Set to ''False''' to avoid the warning.

Registered keywords are handled specially by Robot so that:

- Their arguments are not resolved normally (use `args_to_process` to control that). This basically means not replacing variables or handling escapes.
- They are not stopped by timeouts.
- If there are conflicts with keyword names, these keywords have *lower* precedence than other keywords.

Main use cases are:

- Library keyword is using *BuiltIn.run_keyword* internally to execute other keywords. Registering the caller as a "run keyword variant" avoids variables and escapes in arguments being resolved multiple times. All arguments passed to *run_keyword* can and should be left unresolved.
- Keyword has some need to not resolve variables in arguments. This way variable values are not logged anywhere by Robot automatically.

As mentioned above, this API will likely be reimplemented in the future or there could be new API for library keywords to execute other keywords. External libraries can nevertheless use this API if they really need it and are aware of the possible breaking changes in the future.

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')

from robot.libraries.BuiltIn import BuiltIn, register_run_keyword

class MyLibrary:

def my_run_keyword_if(self, expression, name, *args): # Do something if self._is_true(expression):

return BuiltIn().run_keyword(name, *args)

Process one argument normally to get *expression* resolved. register_run_keyword('MyLibrary', 'my_run_keyword_if', args_to_process=1)

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 extend also with other iterables. *Convert To List* can be used to convert tuples and other iterables to Python list objects.

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

= Boolean arguments =

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

True examples:

False examples:

Considering `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 = '6.0.2'

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

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

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

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

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

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

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

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

See also `Should Not Contain Match`.

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

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

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

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

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

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

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

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

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

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

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

The given lists are not altered by this keyword.

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

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

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

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*, *default=*)
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. If optional `default` value is given, it will be returned instead of failing.

The given dictionary is never altered by this keyword.

Support for `default` is new in Robot Framework 6.0.

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 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 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|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|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`
- `microseconds`, `microsecond`, `us`, `μs` (new in RF 6.0)
- `nanoseconds`, `nanosecond`, `ns` (new in RF 6.0)

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

== Timer string ==

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

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

== Python timedelta ==

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

= Millisecond handling =

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

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

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

= Programmatic usage =

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

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

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

Returns current local or UTC time with an optional increment.

Arguments: - `time_zone`: Get the current time on this time zone. Currently only

`local` (default) and `UTC` are supported. 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'*, *exclude_millis=False*, *date_format=None*)

Converts between supported *date formats*.

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

milliseconds as explained in *millisecond handling*.

- `date_format`: Specifies possible *custom timestamp* format.

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

Converts between supported *time formats*.

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

milliseconds as explained in *millisecond handling*.

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

Subtracts date from another date and returns time between.

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

supported *date formats*.

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

Examples:

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

Adds time to date and returns the resulting date.

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

date formats.

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

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

Subtracts time from date and returns the resulting date.

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

date formats.

- **`time`: Time that is subtracted in one of the supported** *time formats*.

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

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

Adds time to another time and returns the resulting time.

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

milliseconds as explained in *millisecond handling*.

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

Subtracts time from another time and returns the resulting time.

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

the supported *time formats*.

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

robot.libraries.Dialogs module

A 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#truth|rules 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 accept 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)|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|regular expressions] that are more powerful but also more complicated that glob patterns. The regular expression support is implemented using Python's [http://docs.python.org/library/re.html|re module] and its documentation should be consulted for more information about the syntax.

Because the backslash character (`\`) is an escape character in Robot Framework 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.

= `pathlib.Path` support =

Starting from Robot Framework 6.0, arguments representing paths can be given as [https://docs.python.org/3/library/pathlib.html pathlib.Path] instances in addition to strings.

All keywords returning paths return them as strings. This may change in the future so that the return value type matches the argument type.

= Boolean arguments =

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

True examples:

False examples:

= Example =

ROBOT_LIBRARY_SCOPE = 'GLOBAL'

ROBOT_LIBRARY_VERSION = '6.0.2'

run(*command*)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

`encoding` defines the encoding of the file. The default value is `UTF-8`, which means that UTF-8 and ASCII encoded files are read correctly. In addition to the encodings supported by the underlying Python implementation, the following special encoding values can be used:

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

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

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

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.
- Converts `pathlib.Path` instances to `str`.
- ${path1} = 'abc'
- ${path2} = 'def'
- ${path3} = 'abc/def/ghi'
- ${path4} = '/home/robot/stuff'

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

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

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

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

split_extension (*path*)
Splits the extension from the given path.

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

- ${path} = 'file' & ${ext} = 'extension'
- ${p2} = 'path/file' & ${e2} = 'ext'
- ${p3} = 'path/file' & ${e3} = ''
- ${p4} = 'p2/file' & ${e4} = 'ext'
- ${p5} = 'path/.file' & ${e5} = 'ext'

- ${p6} = 'path/.file' & ${e6} = ''

get_modified_time (*path*, *format='timestamp'*)
Returns the last modification time of a file or directory.

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

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

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

set_modified_time (*path*, *mtime*)
Sets the file modification and access times.

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

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

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

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

File and directory names are returned in case-sensitive alphabetical order, e.g. `['A Name', 'Second', 'a lower case name', 'one more']`. Implicit directories `.` and `..` are not returned. The returned items are automatically logged.

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

If `pattern` is given, only items matching it are returned. The pattern 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] module and its [http://docs.python.org/library/subprocess.html#popen-constructor|Popen] class.

The library has following main usages:

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

== Table of contents ==

%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|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 then 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#truth|rules 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 = '6.0.2'

TERMINATE_TIMEOUT = 30

KILL_TIMEOUT = 10

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

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

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

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

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

This keyword does not change the *active process*.

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

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

Makes the started process new *active process*. Returns the created [https://docs.python.org/3/library/subprocess.html#popen-constructor | subprocess.Popen] object which can be 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 finishes before this keyword can be used. In practice this means that processes started with *Start Process* must be finished either with *Wait For Process* or *Terminate Process* before using this keyword.

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

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

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

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

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

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

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-ÿ]')

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*, *screenshot_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 = '6.0.2'

set_screenshot_directory(*path*)
Sets the directory where screenshots are saved.

It is possible to use `/` as a path separator in all operating systems. Path to the old directory is returned.

The directory can also be set in *importing*.

take_screenshot(*name='screenshot'*, *width='800px'*)
Takes a screenshot in JPEG format and embeds it into the log file.

Name of the file where the screenshot is stored is derived from the given `name`. If the `name` ends with extension `.jpg` or `.jpeg`, the screenshot will be stored with that exact name. Otherwise a unique name is created by adding an underscore, a running index and an extension to the `name`.

The name will be interpreted to be relative to the directory where the log file is written. It is also possible to use absolute paths. Using `/` as a path separator works in all operating systems.

`width` specifies the size of the screenshot in the log file.

The path where the screenshot is saved is returned.

take_screenshot_without_embedding(*name='screenshot'*)
Takes a screenshot and links it from the log file.

This keyword is otherwise identical to *Take Screenshot* but the saved screenshot is not embedded into the log file. The screenshot is linked so it is nevertheless easily available.

class robot.libraries.Screenshot.**ScreenshotTaker**(*module_name=None*)
Bases: `object`

test(*path=None*)

robot.libraries.String module

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

A 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 = '6.0.2'

convert_to_lower_case(*string*)
Converts string to lower case.

Uses Python's standard [https://docs.python.org/library/stdtypes.html#str.lower|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|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|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|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*, *flags=None*)
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.

Lines match only if they match the pattern fully by default, 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.

Possible flags altering how the expression is parsed (e.g. `re.IGNORECASE`, `re.VERBOSE`) can be given using the `flags` argument (e.g. `flags=IGNORECASE | VERBOSE`) or embedded to the pattern (e.g. `(?ix)pattern`).

Lines are returned as one string concatenated back together with newlines. Possible trailing newline is never returned. The number of matching lines is automatically logged.

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

The `flags` argument is new in Robot Framework 6.0.

get_regexp_matches (*string*, *pattern*, **groups*, *flags=None*)
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.

Possible flags altering how the expression is parsed (e.g. `re.IGNORECASE`, `re.MULTILINE`) can be given using the `flags` argument (e.g. `flags=IGNORECASE | MULTILINE`) or embedded to the pattern (e.g. `(?im)pattern`).

The `flags` argument is new in Robot Framework 6.0.

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*, *flags=None*)
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.

Possible flags altering how the expression is parsed (e.g. `re.IGNORECASE`, `re.MULTILINE`) can be given using the `flags` argument (e.g. `flags=IGNORECASE | MULTILINE`) or embedded to the pattern (e.g. `(?im)pattern`).

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

The `flags` argument is new in Robot Framework 6.0.

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*, *flags=None*)
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.

Possible flags altering how the expression is parsed (e.g. `re.IGNORECASE`, `re.MULTILINE`) can be given using the `flags` argument (e.g. `flags=IGNORECASE | MULTILINE`) or embedded to the pattern (e.g. `(?im)pattern`).

The `flags` argument is new in Robot Framework 6.0.

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_regexp=False*, *encoding='UTF-8'*, *encoding_errors='ignore'*, *default_log_level='INFO'*, *window_size=None*, *environ_user=None*, *terminal_emulation=False*, *terminal_type=None*, *telnetlib_log_level='TRACE'*, *connection_timeout=None*)

Bases: `object`

A 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.iolPyte]. Terminal emulation will process the output in a virtual screen. This means that ANSI escape codes, like cursor movements, and also control characters, like carriage returns and backspaces, have the same effect on the result as they would have on a normal terminal screen. For example the sequence `acdc\x1b[3Dbba` will result in output `abba`.

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

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

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

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

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

= Logging =

All keywords that read something log the output. These keywords take the log level to use as an optional argument, and if no log level is specified they use the [#Configuration|configured] default value.

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

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

= Time string format =

Timeouts and other times used must be given as a time string using format like `15 seconds` or `1min 10s`. If the timeout is given as just a number, for example, `10` or `1.5`, it is considered to be seconds. The time string format is described in more detail in an appendix of [http://robotframework.org/robotframework/#user-guide|Robot Framework User Guide].

= Boolean arguments =

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

True examples:

False examples:

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

Telnet library can be imported with optional configuration parameters.

Configuration parameters are used as default values when new connections are opened with *Open Connection* keyword. They can also be overridden after opening the connection using the *Set ... keywords*. See these keywords as well as *Configuration*, *Terminal emulation* and *Logging* sections above for more information about these parameters and their possible values.

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

ROBOT_LIBRARY_SCOPE = 'SUITE'

ROBOT_LIBRARY_VERSION = '6.0.2'

get_keyword_names()

open_connection(*host*, *alias=None*, *port=23*, *timeout=None*, *newline=None*, *prompt=None*, *prompt_is_regexp=False*, *encoding=None*, *encoding_errors=None*, *default_log_level=None*, *window_size=None*, *environ_user=None*, *terminal_emulation=None*, *terminal_type=None*, *telnetlib_log_level=None*, *connection_timeout=None*)

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

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

Possible already opened connections are cached and it is possible to switch back to them using *Switch Connection* keyword. It is possible to switch either using explicitly given `alias` or using index returned by this keyword. Indexing starts from 1 and is reset back to it by *Close All Connections* keyword.

switch_connection(*index_or_alias*)

Switches between active connections using an index or an alias.

Aliases can be given to *Open Connection* keyword which also always returns the connection index.

This keyword returns the index of previous active connection.

The example above expects that there were no other open connections when opening the first one, because it used index `1` when switching to the connection later. If you are not sure about that, you can store the index into a variable as shown below.

close_all_connections()

Closes all open connections and empties the connection cache.

If multiple connections are opened, this keyword should be used in a test or suite teardown to make sure that all connections are closed. It is not an error if some of the connections have already been closed by *Close Connection*.

After this keyword, new indexes returned by *Open Connection* keyword are reset to 1.

class robot.libraries.Telnet.**TelnetConnection**(*host=None*, *port=23*, *timeout=3.0*, *newline='CRLF'*, *prompt=None*, *prompt_is_regexp=False*, *encoding='UTF-8'*, *encoding_errors='ignore'*, *default_log_level='INFO'*, *window_size=None*, *environ_user=None*, *terminal_emulation=False*, *terminal_type=None*, *telnetlib_log_level='TRACE'*, *connection_timeout=None*)

Bases: `telnetlib.Telnet`

NEW_ENVIRON_IS = 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 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|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#truth|rules 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)|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 = '6.0.2'

parse_xml(*source*, *keep_clark_notation=False*, *strip_namespaces=False*)
Parses the given XML file or string into an element structure.

The `source` can either be a path to an XML file or a string containing XML. In both cases the XML is parsed into ElementTree [http://docs.python.org/library/xml.etree.elementtree.html#element-objects|element structure] and the root element is returned. Possible comments and processing instructions in the source XML are removed.

As discussed in *Handling XML namespaces* section, this keyword, by default, removes namespace information ElementTree has added to tag names and moves it into `xmlns` attributes. This typically eases handling XML documents with namespaces considerably. If you do not want that to happen, or want to avoid the small overhead of going through the element structure when your XML does not have namespaces, you can disable this feature by giving `keep_clark_notation` argument a true value (see *Boolean arguments*).

If you want to strip namespace information altogether so that it is not included even if XML is saved, you can give a true value to `strip_namespaces` argument.

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 MAXNUMER/MAXDENOM. Return a tuple of the actual values if no argument is given.

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

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

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

On Macintosh, XXXXX

On Unix, there are currently no special attribute values.

bbox (*column=None*, *row=None*, *col2=None*, *row2=None*)
Return a tuple of integer coordinates for the bounding box of this widget controlled by the geometry manager grid.

If COLUMN, ROW is given the bounding box applies from the cell with row and column 0 to the specified cell. If COL2 and ROW2 are given the bounding box starts at that cell.

The returned integers specify the offset of the upper left corner in the master widget and the width and height.

bell (*displayof=0*)
Ring a display's bell.

bind (*sequence=None*, *func=None*, *add=None*)
Bind to this widget at event SEQUENCE a call to function FUNC.

SEQUENCE is a string of concatenated event patterns. An event pattern is of the form <MODIFIER-MODIFIER-TYPE-DETAIL> where MODIFIER is one of Control, Mod2, M2, Shift, Mod3, M3, Lock, Mod4, M4, Button1, B1, Mod5, M5 Button2, B2, Meta, M, Button3, B3, Alt, Button4, B4, Double, Button5, B5 Triple, Mod1, M1. TYPE is one of Activate, Enter, Map, ButtonPress, Button, Expose, Motion, ButtonRelease FocusIn, MouseWheel, Circulate, FocusOut, Property, Colormap, Gravity Reparent, Configure, KeyPress, Key, Unmap, Deactivate, KeyRelease Visibility, Destroy, Leave and DETAIL is the button number for ButtonPress, ButtonRelease and DETAIL is the Keysym for KeyPress and KeyRelease.

Examples are <Control-Button-1> for pressing Control and mouse button 1 or <Alt-A> for pressing A and the Alt key (KeyPress can be omitted). An event pattern can also be a virtual event of the form <<AString>> where AString can be arbitrary. This event can be generated by event_generate. If events are concatenated they must appear shortly after each other.

FUNC will be called if the event sequence occurs with an instance of Event as argument. If the return value of FUNC is "break" no further bound function is invoked.

An additional boolean parameter ADD specifies whether FUNC will be called additionally to the other bound function or whether it will replace the previous function.

Bind will return an identifier to allow deletion of the bound function with unbind without memory leak.

If FUNC or SEQUENCE is omitted the bound function or list of bound events are returned.

bind_all (*sequence=None*, *func=None*, *add=None*)
Bind to all widgets at an event SEQUENCE a call to function FUNC. An additional boolean parameter ADD specifies whether FUNC will be called additionally to the other bound function or whether it will replace the previous function. See bind for the return value.

bind_class (*className*, *sequence=None*, *func=None*, *add=None*)
Bind to widgets with bindtag CLASSNAME at event SEQUENCE a call of function FUNC. An additional boolean parameter ADD specifies whether FUNC will be called additionally to the other bound function or whether it will replace the previous function. See bind for the return value.

bindtags (*tagList=None*)
Set or get the list of bindtags for this widget.

With no argument return the list of all bindtags associated with this widget. With a list of strings as argument the bindtags are set to this list. The bindtags determine in which order events are processed (see bind).

cget (*key*)
Return the resource value for a KEY given as string.

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

clipboard_append (*string*, ***kw*)
Append STRING to the Tk clipboard.

A widget specified at the optional displayof keyword argument specifies the target display. The clipboard can be retrieved with selection_get.

clipboard_clear (***kw*)
Clear the data in the Tk clipboard.

A widget specified for the optional displayof keyword argument specifies the target display.

clipboard_get (***kw*)
Retrieve data from the clipboard on window's display.

The window keyword defaults to the root window of the Tkinter application.

The type keyword specifies the form in which the data is to be returned and should be an atom name such as STRING or FILE_NAME. Type defaults to STRING, except on X11, where the default is to try UTF8_STRING and fall back to STRING.

This command is equivalent to:

selection_get(CLIPBOARD)

colormapwindows (**wlist*)
Store list of window names (WLIST) into WM_COLORMAPWINDOWS property of this widget. This

list contains windows whose colormaps differ from their parents. Return current list of widgets if WLIST is empty.

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

iconify()
Display widget as icon.

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

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

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 X if None is given.

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

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

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

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

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

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

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

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

wm_iconify()
Display widget as icon.

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

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

wm_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 X if None is given.

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

wm_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 <<AString>> where AString can be arbitrary. This event can be generated by event_generate. If events are concatenated they must appear shortly after each other.

FUNC will be called if the event sequence occurs with an instance of Event as argument. If the return value of FUNC is "break" no further bound function is invoked.

An additional boolean parameter ADD specifies whether FUNC will be called additionally to the other bound function or whether it will replace the previous function.

Bind will return an identifier to allow deletion of the bound function with unbind without memory leak.

If FUNC or SEQUENCE is omitted the bound function or list of bound events are returned.

bind_all (*sequence=None*, *func=None*, *add=None*)
Bind to all widgets at an event SEQUENCE a call to function FUNC. An additional boolean parameter ADD specifies whether FUNC will be called additionally to the other bound function or whether it will replace the previous function. See bind for the return value.

bind_class (*className*, *sequence=None*, *func=None*, *add=None*)
Bind to widgets with bindtag CLASSNAME at event SEQUENCE a call of function FUNC. An additional boolean parameter ADD specifies whether FUNC will be called additionally to the other bound function or whether it will replace the previous function. See bind for the return value.

bindtags (*tagList=None*)
Set or get the list of bindtags for this widget.

With no argument return the list of all bindtags associated with this widget. With a list of strings as argument the bindtags are set to this list. The bindtags determine in which order events are processed (see bind).

cget (*key*)
Return the resource value for a KEY given as string.

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

clipboard_append (*string*, ***kw*)
Append STRING to the Tk clipboard.

A widget specified at the optional displayof keyword argument specifies the target display. The clipboard can be retrieved with selection_get.

clipboard_clear (***kw*)
Clear the data in the Tk clipboard.

A widget specified for the optional displayof keyword argument specifies the target display.

clipboard_get (***kw*)
Retrieve data from the clipboard on window's display.

The window keyword defaults to the root window of the Tkinter application.

The type keyword specifies the form in which the data is to be returned and should be an atom name such as STRING or FILE_NAME. Type defaults to STRING, except on X11, where the default is to try UTF8_STRING and fall back to STRING.

This command is equivalent to:

selection_get(CLIPBOARD)

colormapwindows (**wlist*)
Store list of window names (WLIST) into WM_COLORMAPWINDOWS property of this widget. This

list contains windows whose colormaps differ from their parents. Return current list of widgets if WLIST is empty.

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

iconify()
Display widget as icon.

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

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

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 X if None is given.

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

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

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

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

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

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

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

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

wm_iconify()
Display widget as icon.

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

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

wm_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 X if None is given.

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

wm_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 <<AString>> where AString can be arbitrary. This event can be generated by event_generate. If events are concatenated they must appear shortly after each other.

FUNC will be called if the event sequence occurs with an instance of Event as argument. If the return value of FUNC is "break" no further bound function is invoked.

An additional boolean parameter ADD specifies whether FUNC will be called additionally to the other bound function or whether it will replace the previous function.

Bind will return an identifier to allow deletion of the bound function with unbind without memory leak.

If FUNC or SEQUENCE is omitted the bound function or list of bound events are returned.

bind_all (*sequence=None*, *func=None*, *add=None*)
Bind to all widgets at an event SEQUENCE a call to function FUNC. An additional boolean parameter ADD specifies whether FUNC will be called additionally to the other bound function or whether it will replace the previous function. See bind for the return value.

bind_class (*className*, *sequence=None*, *func=None*, *add=None*)
Bind to widgets with bindtag CLASSNAME at event SEQUENCE a call of function FUNC. An additional boolean parameter ADD specifies whether FUNC will be called additionally to the other bound function or whether it will replace the previous function. See bind for the return value.

bindtags (*tagList=None*)
Set or get the list of bindtags for this widget.

With no argument return the list of all bindtags associated with this widget. With a list of strings as argument the bindtags are set to this list. The bindtags determine in which order events are processed (see bind).

cget (*key*)
Return the resource value for a KEY given as string.

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

clipboard_append (*string*, ***kw*)
Append STRING to the Tk clipboard.

A widget specified at the optional displayof keyword argument specifies the target display. The clipboard can be retrieved with selection_get.

clipboard_clear (***kw*)
Clear the data in the Tk clipboard.

A widget specified for the optional displayof keyword argument specifies the target display.

clipboard_get (***kw*)
Retrieve data from the clipboard on window's display.

The window keyword defaults to the root window of the Tkinter application.

The type keyword specifies the form in which the data is to be returned and should be an atom name such as STRING or FILE_NAME. Type defaults to STRING, except on X11, where the default is to try UTF8_STRING and fall back to STRING.

This command is equivalent to:

selection_get(CLIPBOARD)

colormapwindows (**wlist*)
Store list of window names (WLIST) into WM_COLORMAPWINDOWS property of this widget. This

list contains windows whose colormaps differ from their parents. Return current list of widgets if WLIST is empty.

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

iconify()
Display widget as icon.

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

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

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 X if None is given.

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

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

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

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

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

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

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

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

wm_iconify()
Display widget as icon.

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

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

wm_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 X if None is given.

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

wm_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 <<AString>> where AString can be arbitrary. This event can be generated by event_generate. If events are concatenated they must appear shortly after each other.

FUNC will be called if the event sequence occurs with an instance of Event as argument. If the return value of FUNC is "break" no further bound function is invoked.

An additional boolean parameter ADD specifies whether FUNC will be called additionally to the other bound function or whether it will replace the previous function.

Bind will return an identifier to allow deletion of the bound function with unbind without memory leak.

If FUNC or SEQUENCE is omitted the bound function or list of bound events are returned.

bind_all (*sequence=None*, *func=None*, *add=None*)
Bind to all widgets at an event SEQUENCE a call to function FUNC. An additional boolean parameter ADD specifies whether FUNC will be called additionally to the other bound function or whether it will replace the previous function. See bind for the return value.

bind_class (*className*, *sequence=None*, *func=None*, *add=None*)
Bind to widgets with bindtag CLASSNAME at event SEQUENCE a call of function FUNC. An additional boolean parameter ADD specifies whether FUNC will be called additionally to the other bound function or whether it will replace the previous function. See bind for the return value.

bindtags (*tagList=None*)
Set or get the list of bindtags for this widget.

With no argument return the list of all bindtags associated with this widget. With a list of strings as argument the bindtags are set to this list. The bindtags determine in which order events are processed (see bind).

cget (*key*)
Return the resource value for a KEY given as string.

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

clipboard_append (*string*, ***kw*)
Append STRING to the Tk clipboard.

A widget specified at the optional displayof keyword argument specifies the target display. The clipboard can be retrieved with selection_get.

clipboard_clear (***kw*)
Clear the data in the Tk clipboard.

A widget specified for the optional displayof keyword argument specifies the target display.

clipboard_get (***kw*)
Retrieve data from the clipboard on window's display.

The window keyword defaults to the root window of the Tkinter application.

The type keyword specifies the form in which the data is to be returned and should be an atom name such as STRING or FILE_NAME. Type defaults to STRING, except on X11, where the default is to try UTF8_STRING and fall back to STRING.

This command is equivalent to:

selection_get(CLIPBOARD)

colormapwindows (**wlist*)
Store list of window names (WLIST) into WM_COLORMAPWINDOWS property of this widget. This

list contains windows whose colormaps differ from their parents. Return current list of widgets if WLIST is empty.

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

iconify()
Display widget as icon.

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

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

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 X if None is given.

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

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

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

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

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

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

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

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

wm_iconify()
Display widget as icon.

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

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

wm_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 X if None is given.

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

wm_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 <<AString>> where AString can be arbitrary. This event can be generated by event_generate. If events are concatenated they must appear shortly after each other.

FUNC will be called if the event sequence occurs with an instance of Event as argument. If the return value of FUNC is "break" no further bound function is invoked.

An additional boolean parameter ADD specifies whether FUNC will be called additionally to the other bound function or whether it will replace the previous function.

Bind will return an identifier to allow deletion of the bound function with unbind without memory leak.

If FUNC or SEQUENCE is omitted the bound function or list of bound events are returned.

bind_all (*sequence=None*, *func=None*, *add=None*)
Bind to all widgets at an event SEQUENCE a call to function FUNC. An additional boolean parameter ADD specifies whether FUNC will be called additionally to the other bound function or whether it will replace the previous function. See bind for the return value.

bind_class (*className*, *sequence=None*, *func=None*, *add=None*)
Bind to widgets with bindtag CLASSNAME at event SEQUENCE a call of function FUNC. An additional boolean parameter ADD specifies whether FUNC will be called additionally to the other bound function or whether it will replace the previous function. See bind for the return value.

bindtags (*tagList=None*)
Set or get the list of bindtags for this widget.

With no argument return the list of all bindtags associated with this widget. With a list of strings as argument the bindtags are set to this list. The bindtags determine in which order events are processed (see bind).

cget (*key*)
Return the resource value for a KEY given as string.

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

clipboard_append (*string*, ***kw*)
Append STRING to the Tk clipboard.

A widget specified at the optional displayof keyword argument specifies the target display. The clipboard can be retrieved with selection_get.

clipboard_clear (***kw*)
Clear the data in the Tk clipboard.

A widget specified for the optional displayof keyword argument specifies the target display.

clipboard_get (***kw*)
Retrieve data from the clipboard on window's display.

The window keyword defaults to the root window of the Tkinter application.

The type keyword specifies the form in which the data is to be returned and should be an atom name such as STRING or FILE_NAME. Type defaults to STRING, except on X11, where the default is to try UTF8_STRING and fall back to STRING.

This command is equivalent to:

selection_get(CLIPBOARD)

colormapwindows (**wlist*)
Store list of window names (WLIST) into WM_COLORMAPWINDOWS property of this widget. This

list contains windows whose colormaps differ from their parents. Return current list of widgets if WLIST is empty.

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

iconify()
Display widget as icon.

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

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

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 X if None is given.

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

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

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

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

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

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

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

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

wm_iconify()
Display widget as icon.

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

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

wm_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 X if None is given.

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

wm_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*, *metadata=None*, *set_tags=None*, *include_tags=None*, *exclude_tags=None*, *include_suites=None*, *include_tests=None*, *empty_suite_ok=False*)
Bases: *robot.model.visitor.SuiteVisitor*

add_tags

remove_tags

visit_suite(*suite*)
Implements traversing through 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 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.**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*, *include_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=()*, *timeout=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 Fr

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*, *values_as_strings=False*, *html_escape=False*)

total

add_test(*test*)

visit(*visitor*)

class robot.model.stats.**TotalStat**(*name*)
Bases: *robot.model.stats.Stat*

Stores statistic values for a test run.

type = 'total'

add_test(*test*)

get_attributes(*include_label=False*, *include_elapsed=False*, *exclude_empty=True*, *values_as_strings=False*, *html_escape=False*)

total

visit(*visitor*)

class robot.model.stats.**SuiteStat**(*suite*)
Bases: *robot.model.stats.Stat*

Stores statistics values for a single suite.

type = 'suite'

id = None
Identifier of the suite, e.g. *s1-s2*.

elapsed = None
Number of milliseconds it took to execute this suite, including sub-suites.

add_stat(*other*)

add_test(*test*)

get_attributes(*include_label=False*, *include_elapsed=False*, *exclude_empty=True*, *values_as_strings=False*, *html_escape=False*)

total

visit(*visitor*)

class robot.model.stats.**TagStat**(*name*, *doc=''*, *links=None*, *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.suitestatistics module

class robot.model.suitestatistics.**SuiteStatistics**(*suite*)
Bases: `object`

Container for suite statistics.

stat = None
Instance of *SuiteStat*.

suites = None
List of *TestSuite* objects.

visit(*visitor*)

class robot.model.suitestatistics.**SuiteStatisticsBuilder**(*suite_stat_level*)
Bases: `object`

current

start_suite(*suite*)

add_test(*test*)

end_suite()

robot.model.tags module

class robot.model.tags.**Tags**(*tags=None*)
Bases: object

robot(*name*)
Check do tags contain a special tag in format *robot:<name>*.

This is same as *'robot:<name>' in tags* but considerably faster.

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 about 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

class robot.model.testsuite.**TestSuite**(*name=''*, *doc=''*, *metadata=None*, *source=None*, *rpa=False*, *parent=None*)
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 about 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


```

(continues on next page)

(continued from previous page)

```
class SelectEveryXthTest(SuiteVisitor):

    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*, *colors='AUTO'*)
Bases: `object`

write(*text*, *flush=True*)

flush()

highlight(*text*, *status=None*, *flush=True*)

error(*message*, *level*)

robot.output.console.highlighting.**Highlighter**(*stream*)

class robot.output.console.highlighting.**AnsiHighlighter**(*stream*)
Bases: `object`

green()

red()

yellow()

reset()

class robot.output.console.highlighting.**NoHighlighting**(*stream*)
Bases: *robot.output.console.highlighting.AnsiHighlighter*

green()

red()

reset()

yellow()

class robot.output.console.highlighting.**DosHighlighter**(*stream*)
Bases: `object`

green()

red()

yellow()

reset()

robot.output.console.quiet module

class robot.output.console.quiet.**QuietOutput**(*colors='AUTO'*, *stderr=None*)
Bases: object

message(*msg*)

class robot.output.console.quiet.**NoOutput**
Bases: object

robot.output.console.verbose module

class robot.output.console.verbose.**VerboseOutput**(*width=78*, *colors='AUTO'*, *markers='AUTO'*, *stdout=None*, *stderr=None*)
Bases: object

start_suite(*suite*)

end_suite(*suite*)

start_test(*test*)

end_test(*test*)

start_keyword(*kw*)

end_keyword(*kw*)

message(*msg*)

output_file(*name*, *path*)

class robot.output.console.verbose.**VerboseWriter**(*width=78*, *colors='AUTO'*, *markers='AUTO'*, *stdout=None*, *stderr=None*)
Bases: object

info(*name*, *doc*, *start_suite=False*)

suite_separator()

test_separator()

status(*status*, *clear=False*)

message(*message*)

keyword_marker(*status*)

error(*message*, *level*, *clear=False*)

output(*name*, *path*)

class robot.output.console.verbose.**KeywordMarker**(*highlighter*, *markers*)
Bases: object

mark(*status*)

reset_count()

Submodules

robot.output.debugfile module

robot.output.debugfile.**DebugFile**(*path*)

robot.output.filelogger module

class robot.output.filelogger.**FileLogger**(*path*, *level*)

Bases: *robot.output.loggerhelper.AbstractLogger*

message(*msg*)

start_suite(*suite*)

end_suite(*suite*)

start_test(*test*)

end_test(*test*)

start_keyword(*kw*)

end_keyword(*kw*)

output_file(*name*, *path*)

close()

debug(*msg*)

error(*msg*)

fail(*msg*)

info(*msg*)

set_level(*level*)

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*, *timestamp=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/v6.0.2/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.statementlexers.Lexer*

accepts_more(*statement*)

input(*statement*)

lexer_for(*statement*)

lexer_classes()

lex()

classmethod handles(*statement*, *ctx*)

class robot.parsing.lexer.blocklexers.**FileLexer**(*ctx*)
Bases: *robot.parsing.lexer.blocklexers.BlockLexer*

lex()

lexer_classes()

accepts_more(*statement*)

classmethod handles(*statement*, *ctx*)

input(*statement*)

lexer_for(*statement*)

class robot.parsing.lexer.blocklexers.**SectionLexer**(*ctx*)
Bases: *robot.parsing.lexer.blocklexers.BlockLexer*

accepts_more(*statement*)

classmethod handles(*statement*, *ctx*)

input(*statement*)

lex()

lexer_classes()

lexer_for(*statement*)

class robot.parsing.lexer.blocklexers.**SettingSectionLexer**(*ctx*)
Bases: *robot.parsing.lexer.blocklexers.SectionLexer*

classmethod handles(*statement*, *ctx*)

lexer_classes()

accepts_more(*statement*)

input(*statement*)

lex()

lexer_for(*statement*)

class robot.parsing.lexer.blocklexers.**VariableSectionLexer**(*ctx*)
Bases: *robot.parsing.lexer.blocklexers.SectionLexer*

classmethod handles(*statement*, *ctx*)

lexer_classes()

accepts_more(*statement*)

input(*statement*)

lex()

lexer_for(*statement*)

class robot.parsing.lexer.blocklexers.**TestCaseSectionLexer**(*ctx*)
Bases: *robot.parsing.lexer.blocklexers.SectionLexer*

classmethod handles(*statement*, *ctx*)

lexer_classes()

accepts_more(*statement*)

input(*statement*)

lex()

lexer_for(*statement*)

class robot.parsing.lexer.blocklexers.**TaskSectionLexer**(*ctx*)
Bases: *robot.parsing.lexer.blocklexers.SectionLexer*

classmethod handles(*statement*, *ctx*)

lexer_classes()

accepts_more(*statement*)

input(*statement*)

lex()

lexer_for(*statement*)

class robot.parsing.lexer.blocklexers.**KeywordSectionLexer**(*ctx*)
Bases: *robot.parsing.lexer.blocklexers.SettingSectionLexer*

classmethod handles(*statement*, *ctx*)

lexer_classes()

accepts_more(*statement*)

input(*statement*)

lex()

lexer_for(*statement*)

class robot.parsing.lexer.blocklexers.**CommentSectionLexer**(*ctx*)
Bases: *robot.parsing.lexer.blocklexers.SectionLexer*

classmethod handles(*statement*, *ctx*)

lexer_classes()

accepts_more(*statement*)

input(*statement*)

lex()

lexer_for(*statement*)

class robot.parsing.lexer.blocklexers.**ImplicitCommentSectionLexer**(*ctx*)
Bases: *robot.parsing.lexer.blocklexers.SectionLexer*

classmethod handles(*statement*, *ctx*)

lexer_classes()

accepts_more(*statement*)

input(*statement*)

lex()

lexer_for(*statement*)

class robot.parsing.lexer.blocklexers.**ErrorSectionLexer**(*ctx*)
Bases: *robot.parsing.lexer.blocklexers.SectionLexer*

classmethod handles(*statement*, *ctx*)

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()

classmethod handles(*statement*, *ctx*)

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*)

classmethod handles(*statement*, *ctx*)

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*)

classmethod handles(*statement*, *ctx*)

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*)

classmethod handles(*statement*, *ctx*)

lex()

lexer_classes()

lexer_for(*statement*)

class robot.parsing.lexer.blocklexers.**ForLexer**(*ctx*)
Bases: *robot.parsing.lexer.blocklexers.NestedBlockLexer*

classmethod handles(*statement*, *ctx*)

lexer_classes()

accepts_more(*statement*)

input(*statement*)

lex()

lexer_for(*statement*)

class robot.parsing.lexer.blocklexers.**WhileLexer**(*ctx*)
Bases: *robot.parsing.lexer.blocklexers.NestedBlockLexer*

classmethod handles(*statement*, *ctx*)

lexer_classes()

accepts_more(*statement*)

input(*statement*)

lex()

lexer_for(*statement*)

class robot.parsing.lexer.blocklexers.**IfLexer**(*ctx*)
Bases: *robot.parsing.lexer.blocklexers.NestedBlockLexer*

classmethod handles(*statement*, *ctx*)

lexer_classes()

accepts_more(*statement*)

input(*statement*)

lex()

lexer_for(*statement*)

class robot.parsing.lexer.blocklexers.**InlineIfLexer**(*ctx*)
Bases: *robot.parsing.lexer.blocklexers.BlockLexer*

classmethod handles(*statement*, *ctx*)

accepts_more(*statement*)

lexer_classes()

input(*statement*)

lex()

lexer_for(*statement*)

class robot.parsing.lexer.blocklexers.**TryLexer**(*ctx*)
Bases: *robot.parsing.lexer.blocklexers.NestedBlockLexer*

classmethod handles(*statement*, *ctx*)

lexer_classes()

accepts_more(*statement*)

input(*statement*)

lex()

lexer_for(*statement*)

robot.parsing.lexer.context module

class robot.parsing.lexer.context.**LexingContext**(*settings=None*, *lang=None*)
Bases: object

settings_class = None

lex_setting(*statement*)

class robot.parsing.lexer.context.**FileContext**(*settings=None*, *lang=None*)
Bases: *robot.parsing.lexer.context.LexingContext*

add_language(*lang*)

keyword_context()

setting_section(*statement*)

variable_section(*statement*)

test_case_section(*statement*)

task_section(*statement*)

keyword_section(*statement*)

comment_section(*statement*)

lex_invalid_section(*statement*)

lex_setting(*statement*)

settings_class = None

class robot.parsing.lexer.context.**TestCaseFileContext**(*settings=None*, *lang=None*)
Bases: *robot.parsing.lexer.context.FileContext*

settings_class
alias of *robot.parsing.lexer.settings.TestCaseFileSettings*

test_case_context()

test_case_section(*statement*)

task_section(*statement*)

add_language(*lang*)

comment_section(*statement*)

keyword_context()

keyword_section(*statement*)

lex_invalid_section(*statement*)

lex_setting(*statement*)

setting_section(*statement*)

variable_section(*statement*)

class robot.parsing.lexer.context.**ResourceFileContext**(*settings=None*, *lang=None*)
Bases: *robot.parsing.lexer.context.FileContext*

settings_class
alias of *robot.parsing.lexer.settings.ResourceFileSettings*

add_language(*lang*)

comment_section(*statement*)

keyword_context()

keyword_section(*statement*)

lex_invalid_section(*statement*)

lex_setting(*statement*)

setting_section(*statement*)

task_section(*statement*)

test_case_section(*statement*)

variable_section(*statement*)

class robot.parsing.lexer.context.**InitFileContext**(*settings=None*, *lang=None*)
Bases: *robot.parsing.lexer.context.FileContext*

settings_class
alias of *robot.parsing.lexer.settings.InitFileSettings*

add_language(*lang*)

comment_section(*statement*)

keyword_context()

keyword_section(*statement*)

lex_invalid_section(*statement*)

lex_setting(*statement*)

setting_section(*statement*)

task_section(*statement*)

test_case_section(*statement*)

variable_section(*statement*)

class robot.parsing.lexer.context.**TestCaseContext**(*settings=None*, *lang=None*)
Bases: *robot.parsing.lexer.context.LexingContext*

template_set

lex_setting(*statement*)

settings_class = None

class robot.parsing.lexer.context.**KeywordContext**(*settings=None*, *lang=None*)
Bases: *robot.parsing.lexer.context.LexingContext*

lex_setting(*statement*)

settings_class = None

template_set

robot.parsing.lexer.lexer module

robot.parsing.lexer.lexer.**get_tokens**(*source*, *data_only=False*, *tokenize_variables=False*, *lang=None*)

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 date 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.
- **lang** – Additional languages to be supported during parsing. Can be a string matching any of the supported language codes or names, an initialized *Language* subclass, a list containing such strings or instances, or a *Languages* instance.

Returns a generator that yields *Token* instances.

robot.parsing.lexer.lexer.**get_resource_tokens**(*source*, *data_only=False*, *tokenize_variables=False*, *lang=None*)

Parses the given source to resource file tokens.

Same as *get_tokens()* otherwise, 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*, *lang=None*)

Parses the given source to init file tokens.

Same as *get_tokens()* otherwise, 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.settings module

class robot.parsing.lexer.settings.**Settings**(*languages*)
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**(*languages*)
Bases: *robot.parsing.lexer.settings.Settings*

names = ('Documentation', 'Metadata', 'Suite Setup', 'Suite Teardown', 'Test Setup', '

aliases = {'Force Tags': 'Test Tags', 'Task Setup': 'Test Setup', 'Task Tags': 'Tes

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**(*languages*)
Bases: *robot.parsing.lexer.settings.Settings*

names = ('Documentation', 'Metadata', 'Suite Setup', 'Suite Teardown', 'Test Setup', '

aliases = {'Force Tags': 'Test Tags', 'Task Setup': 'Test Setup', 'Task Tags': 'Tes

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**(*languages*)
Bases: *robot.parsing.lexer.settings.Settings*

names = ('Documentation', 'Keyword Tags', '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*, *languages*)
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**(*languages*)
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.

classmethod handles(*statement*, *ctx*)

accepts_more(*statement*)

input(*statement*)

lex()

class robot.parsing.lexer.statementlexers.**StatementLexer**(*ctx*)
Bases: *robot.parsing.lexer.statementlexers.Lexer*

token_type = None

accepts_more(*statement*)

input(*statement*)

lex()

classmethod handles(*statement*, *ctx*)

class robot.parsing.lexer.statementlexers.**SingleType**(*ctx*)
Bases: *robot.parsing.lexer.statementlexers.StatementLexer*

lex()

accepts_more(*statement*)

classmethod handles(*statement*, *ctx*)

input(*statement*)

token_type = None

class robot.parsing.lexer.statementlexers.**TypeAndArguments**(*ctx*)
Bases: *robot.parsing.lexer.statementlexers.StatementLexer*

lex()

accepts_more(*statement*)

classmethod handles(*statement*, *ctx*)

input(*statement*)

token_type = None

class robot.parsing.lexer.statementlexers.**SectionHeaderLexer**(*ctx*)
Bases: *robot.parsing.lexer.statementlexers.SingleType*

classmethod handles(*statement*, *ctx*)

accepts_more(*statement*)

input(*statement*)

lex()

token_type = None

class robot.parsing.lexer.statementlexers.**SettingSectionHeaderLexer**(*ctx*)
Bases: *robot.parsing.lexer.statementlexers.SectionHeaderLexer*

token_type = 'SETTING HEADER'

accepts_more(*statement*)

classmethod handles(*statement*, *ctx*)

input(*statement*)

lex()

class robot.parsing.lexer.statementlexers.**VariableSectionHeaderLexer**(*ctx*)
Bases: *robot.parsing.lexer.statementlexers.SectionHeaderLexer*

token_type = 'VARIABLE HEADER'

accepts_more(*statement*)

classmethod handles(*statement*, *ctx*)

input(*statement*)

lex()

class robot.parsing.lexer.statementlexers.**TestCaseSectionHeaderLexer**(*ctx*)
Bases: *robot.parsing.lexer.statementlexers.SectionHeaderLexer*

token_type = 'TESTCASE HEADER'

accepts_more(*statement*)

classmethod handles(*statement*, *ctx*)

input(*statement*)

lex()

class robot.parsing.lexer.statementlexers.**TaskSectionHeaderLexer**(*ctx*)
Bases: *robot.parsing.lexer.statementlexers.SectionHeaderLexer*

token_type = 'TASK HEADER'

accepts_more(*statement*)

classmethod handles(*statement*, *ctx*)

input(*statement*)

lex()

class robot.parsing.lexer.statementlexers.**KeywordSectionHeaderLexer**(*ctx*)
Bases: *robot.parsing.lexer.statementlexers.SectionHeaderLexer*

token_type = 'KEYWORD HEADER'

accepts_more(*statement*)

classmethod handles(*statement*, *ctx*)

input(*statement*)

lex()

class robot.parsing.lexer.statementlexers.**CommentSectionHeaderLexer**(*ctx*)
Bases: *robot.parsing.lexer.statementlexers.SectionHeaderLexer*

token_type = 'COMMENT HEADER'

accepts_more(*statement*)

classmethod handles(*statement*, *ctx*)

input(*statement*)

lex()

class robot.parsing.lexer.statementlexers.**ErrorSectionHeaderLexer**(*ctx*)
Bases: *robot.parsing.lexer.statementlexers.SectionHeaderLexer*

lex()

accepts_more(*statement*)

classmethod handles(*statement*, *ctx*)

input(*statement*)

token_type = None

class robot.parsing.lexer.statementlexers.**CommentLexer**(*ctx*)
Bases: *robot.parsing.lexer.statementlexers.SingleType*

token_type = 'COMMENT'

accepts_more(*statement*)

classmethod handles(*statement*, *ctx*)

input(*statement*)

lex()

class robot.parsing.lexer.statementlexers.**ImplicitCommentLexer**(*ctx*)
Bases: *robot.parsing.lexer.statementlexers.CommentLexer*

input(*statement*)

lex()

accepts_more(*statement*)

classmethod handles(*statement*, *ctx*)

token_type = 'COMMENT'

class robot.parsing.lexer.statementlexers.**SettingLexer**(*ctx*)
Bases: *robot.parsing.lexer.statementlexers.StatementLexer*

lex()

accepts_more(*statement*)

classmethod handles(*statement*, *ctx*)

input(*statement*)

token_type = None

class robot.parsing.lexer.statementlexers.**TestOrKeywordSettingLexer**(*ctx*)
Bases: *robot.parsing.lexer.statementlexers.SettingLexer*

classmethod handles(*statement*, *ctx*)

accepts_more(*statement*)

input(*statement*)

lex()

token_type = None

class robot.parsing.lexer.statementlexers.**VariableLexer**(*ctx*)
Bases: *robot.parsing.lexer.statementlexers.TypeAndArguments*

token_type = 'VARIABLE'

accepts_more(*statement*)

classmethod handles(*statement*, *ctx*)

input(*statement*)

lex()

class robot.parsing.lexer.statementlexers.**KeywordCallLexer**(*ctx*)
Bases: *robot.parsing.lexer.statementlexers.StatementLexer*

lex()

accepts_more(*statement*)

classmethod handles(*statement*, *ctx*)

input(*statement*)

token_type = None

class robot.parsing.lexer.statementlexers.**ForHeaderLexer**(*ctx*)
Bases: *robot.parsing.lexer.statementlexers.StatementLexer*

separators = ('IN', 'IN RANGE', 'IN ENUMERATE', 'IN ZIP')

classmethod handles(*statement*, *ctx*)

lex()

accepts_more(*statement*)

input(*statement*)

token_type = None

class robot.parsing.lexer.statementlexers.**IfHeaderLexer**(*ctx*)
Bases: *robot.parsing.lexer.statementlexers.TypeAndArguments*

token_type = 'IF'

classmethod handles(*statement*, *ctx*)

accepts_more(*statement*)

input(*statement*)

lex()

class robot.parsing.lexer.statementlexers.**InlineIfHeaderLexer**(*ctx*)
Bases: *robot.parsing.lexer.statementlexers.StatementLexer*

token_type = 'INLINE IF'

classmethod handles(*statement*, *ctx*)

lex()

accepts_more(*statement*)

input(*statement*)

class robot.parsing.lexer.statementlexers.**ElseIfHeaderLexer**(*ctx*)
Bases: *robot.parsing.lexer.statementlexers.TypeAndArguments*

token_type = 'ELSE IF'

classmethod handles(*statement*, *ctx*)

accepts_more(*statement*)

input(*statement*)

lex()

class robot.parsing.lexer.statementlexers.**ElseHeaderLexer**(*ctx*)
Bases: *robot.parsing.lexer.statementlexers.TypeAndArguments*

token_type = 'ELSE'

classmethod handles(*statement*, *ctx*)

accepts_more(*statement*)

input(*statement*)

lex()

class robot.parsing.lexer.statementlexers.**TryHeaderLexer**(*ctx*)
Bases: *robot.parsing.lexer.statementlexers.TypeAndArguments*

token_type = 'TRY'

classmethod handles(*statement*, *ctx*)

accepts_more(*statement*)

input(*statement*)

lex()

class robot.parsing.lexer.statementlexers.**ExceptHeaderLexer**(*ctx*)
Bases: *robot.parsing.lexer.statementlexers.StatementLexer*

token_type = 'EXCEPT'

classmethod handles(*statement*, *ctx*)

lex()

accepts_more(*statement*)

input(*statement*)

class robot.parsing.lexer.statementlexers.**FinallyHeaderLexer**(*ctx*)
Bases: *robot.parsing.lexer.statementlexers.TypeAndArguments*

token_type = 'FINALLY'

classmethod handles(*statement*, *ctx*)

accepts_more(*statement*)

input(*statement*)

lex()

class robot.parsing.lexer.statementlexers.**WhileHeaderLexer**(*ctx*)
Bases: *robot.parsing.lexer.statementlexers.StatementLexer*

token_type = 'WHILE'

classmethod handles(*statement*, *ctx*)

lex()

accepts_more(*statement*)

input(*statement*)

class robot.parsing.lexer.statementlexers.**EndLexer**(*ctx*)
Bases: *robot.parsing.lexer.statementlexers.TypeAndArguments*

token_type = 'END'

classmethod handles(*statement*, *ctx*)

accepts_more(*statement*)

input(*statement*)

lex()

class robot.parsing.lexer.statementlexers.**ReturnLexer**(*ctx*)
Bases: *robot.parsing.lexer.statementlexers.TypeAndArguments*

token_type = 'RETURN STATEMENT'

classmethod handles(*statement*, *ctx*)

accepts_more(*statement*)

input(*statement*)

lex()

class robot.parsing.lexer.statementlexers.**ContinueLexer**(*ctx*)
Bases: *robot.parsing.lexer.statementlexers.TypeAndArguments*

token_type = 'CONTINUE'

classmethod handles(*statement*, *ctx*)

accepts_more(*statement*)

input(*statement*)

lex()

class robot.parsing.lexer.statementlexers.**BreakLexer**(*ctx*)
Bases: *robot.parsing.lexer.statementlexers.TypeAndArguments*

token_type = 'BREAK'

classmethod handles(*statement*, *ctx*)

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'
    TASK_HEADER = 'TASK 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'
    KEYWORD_TAGS = 'KEYWORD 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'

CONFIG = 'CONFIG'

EOL = 'EOL'

EOS = 'EOS'

ERROR = 'ERROR'

FATAL_ERROR = 'FATAL ERROR'

NON_DATA_TOKENS = frozenset({'CONTINUATION', 'EOL', 'EOS', 'COMMENT', 'SEPARATOR'})

SETTING_TOKENS = frozenset({'RETURN', 'DOCUMENTATION', 'TEARDOWN', 'TEMPLATE', 'TEST T

HEADER_TOKENS = frozenset({'TESTCASE HEADER', 'COMMENT HEADER', 'VARIABLE HEADER', 'SE

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'

CONFIG = 'CONFIG'

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({'TESTCASE HEADER', 'COMMENT HEADER', 'VARIABLE HEADER', 'SE

IF = 'IF'

INLINE_IF = 'INLINE IF'

```
KEYWORD = 'KEYWORD'
KEYWORD_HEADER = 'KEYWORD HEADER'
KEYWORD_NAME = 'KEYWORD NAME'
KEYWORD_TAGS = 'KEYWORD TAGS'
LIBRARY = 'LIBRARY'
METADATA = 'METADATA'
NAME = 'NAME'
NON_DATA_TOKENS = frozenset({'CONTINUATION', 'EOL', 'EOS', 'COMMENT', 'SEPARATOR'})
OPTION = 'OPTION'
RESOURCE = 'RESOURCE'
RETURN = 'RETURN'
RETURN_SETTING = 'RETURN'
RETURN_STATEMENT = 'RETURN STATEMENT'
SEPARATOR = 'SEPARATOR'
SETTING_HEADER = 'SETTING HEADER'
SETTING_TOKENS = frozenset({'RETURN', 'DOCUMENTATION', 'TEARDOWN', 'TEMPLATE', 'TEST T
SETUP = 'SETUP'
SUITE_SETUP = 'SUITE SETUP'
SUITE_TEARDOWN = 'SUITE TEARDOWN'
TAGS = 'TAGS'
TASK_HEADER = 'TASK HEADER'
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'

CONFIG = 'CONFIG'

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({'TESTCASE HEADER', 'COMMENT HEADER', 'VARIABLE HEADER', 'SE
    IF = 'IF'
    INLINE_IF = 'INLINE IF'
    KEYWORD = 'KEYWORD'
    KEYWORD_HEADER = 'KEYWORD HEADER'
    KEYWORD_NAME = 'KEYWORD NAME'
    KEYWORD_TAGS = 'KEYWORD TAGS'
    LIBRARY = 'LIBRARY'
    METADATA = 'METADATA'
    NAME = 'NAME'
    NON_DATA_TOKENS = frozenset({'CONTINUATION', 'EOL', 'EOS', 'COMMENT', 'SEPARATOR'})
    OPTION = 'OPTION'
    RESOURCE = 'RESOURCE'
    RETURN = 'RETURN'
    RETURN_SETTING = 'RETURN'
    RETURN_STATEMENT = 'RETURN STATEMENT'
    SEPARATOR = 'SEPARATOR'
    SETTING_HEADER = 'SETTING HEADER'
    SETTING_TOKENS = frozenset({'RETURN', 'DOCUMENTATION', 'TEARDOWN', 'TEMPLATE', 'TEST T
    SETUP = 'SETUP'
    SUITE_SETUP = 'SUITE SETUP'
    SUITE_TEARDOWN = 'SUITE TEARDOWN'
    TAGS = 'TAGS'
    TASK_HEADER = 'TASK HEADER'
    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*, *languages=()*)
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*, *orelse=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', 'TASK HEADER'

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.**KeywordTags**(*tokens*, *errors=()*)
Bases: *robot.parsing.model.statements.MultiValue*

type = 'KEYWORD 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'*)
`value` can be given either as a string or as a list of strings.

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=()*, *type=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.**Config**(*tokens*, *errors=()*)
Bases: *robot.parsing.model.statements.Statement*

type = 'CONFIG'

classmethod from_params(*config*, *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'`.

language

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({'KEYWORD HEADER', 'COMMENT HEADER', 'VARIABLE HEADER', ''

handles(*statement*)

parse(*statement*)

class robot.parsing.parser.blockparsers.**TestCaseParser**(*header*)
Bases: *robot.parsing.parser.blockparsers.BlockParser*

handles(*statement*)

parse(*statement*)

unhandled_tokens = frozenset({'KEYWORD HEADER', 'COMMENT HEADER', 'VARIABLE HEADER', ''

class robot.parsing.parser.blockparsers.**KeywordParser**(*header*)
Bases: *robot.parsing.parser.blockparsers.BlockParser*

handles(*statement*)

parse(*statement*)

unhandled_tokens = frozenset({'KEYWORD HEADER', 'COMMENT HEADER', 'VARIABLE HEADER', '

class robot.parsing.parser.blockparsers.**NestedBlockParser**(*model*)
Bases: *robot.parsing.parser.blockparsers.BlockParser*

handles(*statement*)

parse(*statement*)

unhandled_tokens = frozenset({'KEYWORD HEADER', 'COMMENT HEADER', 'VARIABLE HEADER', '

class robot.parsing.parser.blockparsers.**ForParser**(*header*)
Bases: *robot.parsing.parser.blockparsers.NestedBlockParser*

handles(*statement*)

parse(*statement*)

unhandled_tokens = frozenset({'KEYWORD HEADER', 'COMMENT HEADER', 'VARIABLE HEADER', '

class robot.parsing.parser.blockparsers.**IfParser**(*header*, *handle_end=True*)
Bases: *robot.parsing.parser.blockparsers.NestedBlockParser*

parse(*statement*)

handles(*statement*)

unhandled_tokens = frozenset({'KEYWORD HEADER', 'COMMENT HEADER', 'VARIABLE HEADER', '

class robot.parsing.parser.blockparsers.**TryParser**(*header*, *handle_end=True*)
Bases: *robot.parsing.parser.blockparsers.NestedBlockParser*

parse(*statement*)

handles(*statement*)

unhandled_tokens = frozenset({'KEYWORD HEADER', 'COMMENT HEADER', 'VARIABLE HEADER', '

class robot.parsing.parser.blockparsers.**WhileParser**(*header*)
Bases: *robot.parsing.parser.blockparsers.NestedBlockParser*

handles(*statement*)

parse(*statement*)

unhandled_tokens = frozenset({'KEYWORD HEADER', 'COMMENT HEADER', 'VARIABLE HEADER', '

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*, *lang=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 date 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.
- **lang** – Additional languages to be supported during parsing. Can be a string matching any of the supported language codes or names, an initialized *Language* subsclass, a list containing such strings or instances, or a *Languages* instance.

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*, *lang=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*, *lang=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.

class robot.parsing.parser.parser.**SetLanguages**(*file*)

Bases: *robot.parsing.model.visitor.ModelVisitor*

visit_Config(*node*)

generic_visit(*node*)
Called if no explicit visitor function exists for a node.

visit(*node*)
Visit a node.

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*, *expand_keywords=None*)

Bases: object

remove_data_not_needed_in_report()

robot.reporting.jsmodelbuilders module

class robot.reporting.jsmodelbuilders.**JsModelBuilder**(*log_path=None*, *split_log=False*, *expand_keywords=None*, *prune_input_to_save_memory=False*)

Bases: object

build_from(*result_from_xml*)

class robot.reporting.jsmodelbuilders.**SuiteBuilder**(*context*)

Bases: robot.reporting.jsmodelbuilders._Builder

build(*suite*)

class robot.reporting.jsmodelbuilders.**TestBuilder**(*context*)

Bases: robot.reporting.jsmodelbuilders._Builder

build(*test*)

class robot.reporting.jsmodelbuilders.**KeywordBuilder**(*context*)

Bases: robot.reporting.jsmodelbuilders._Builder

build(*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.Branches*

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 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

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*, *endtime=None*, *doc=''*, *parent=None*)
Bases: *robot.model.body.BodyItem*, *robot.result.model.StatusMixin*, *robot.result.modeldeprecation.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'*, *starttime=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.modeldeprecation.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.modeldeprecation.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.modeldeprecation.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 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 IF/ELSE 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.

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.modeldeprecation.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 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
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.modeldeprecation.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.modeldeprecation.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`.

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.

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.modeldeprecation.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.modeldeprecation.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 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 = '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.modeldeprecation.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*, *sourcename=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*, *endtime=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 about 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 about 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*, *include_keywords=True*, *flattened_keywords=None*)
Bases: object

Builds *Result* objects based on output files.

Instead of using this builder directly, it is recommended to use the *ExecutionResult()* factory method.

Parameters

- **source** – Path to the XML output file to build *Result* objects from.

- **include_keywords** – Boolean controlling whether to include keyword information in the result or not. Keywords are not needed when generating only report. Although the 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.xmlelementhandlers module

class robot.result.xmlelementhandlers.**XmlElementHandler**(*execution_result*, *root_handler=None*)

Bases: object

start(*elem*)

end(*elem*)

class robot.result.xmlelementhandlers.**ElementHandler**

Bases: object

element_handlers = {'arg': <robot.result.xmlelementhandlers.ArgumentHandler object>,

tag = None

children = frozenset()

classmethod register(*handler*)

get_child_handler(*tag*)

start(*elem*, *result*)

end(*elem*, *result*)

class robot.result.xmlelementhandlers.**RootHandler**

Bases: *robot.result.xmlelementhandlers.ElementHandler*

children = frozenset({'robot'})

element_handlers = {'arg': <robot.result.xmlelementhandlers.ArgumentHandler object>,

end(*elem*, *result*)

get_child_handler(*tag*)

classmethod register(*handler*)

start(*elem*, *result*)

tag = None

class robot.result.xmlelementhandlers.**RobotHandler**

Bases: *robot.result.xmlelementhandlers.ElementHandler*

tag = 'robot'

children = frozenset({'errors', 'suite', 'statistics'})

start(*elem*, *result*)

element_handlers = {'arg': <robot.result.xmlelementhandlers.ArgumentHandler object>,

end(*elem*, *result*)

get_child_handler(*tag*)

classmethod register(*handler*)

class robot.result.xmlelementhandlers.**SuiteHandler**

Bases: *robot.result.xmlelementhandlers.ElementHandler*

tag = 'suite'

children = frozenset({'doc', 'suite', 'test', 'metadata', 'status', 'meta', 'kw'})

start(*elem*, *result*)

get_child_handler(*tag*)

element_handlers = {'arg': <robot.result.xmlelementhandlers.ArgumentHandler object>,

end(*elem*, *result*)

classmethod register(*handler*)

class robot.result.xmlelementhandlers.**TestHandler**
Bases: *robot.result.xmlelementhandlers.ElementHandler*

tag = 'test'

children = frozenset({'doc', 'timeout', 'if', 'continue', 'return', 'msg', 'tag', 'tag

start(*elem*, *result*)

element_handlers = {'arg': <robot.result.xmlelementhandlers.ArgumentHandler object>,

end(*elem*, *result*)

get_child_handler(*tag*)

classmethod register(*handler*)

class robot.result.xmlelementhandlers.**KeywordHandler**
Bases: *robot.result.xmlelementhandlers.ElementHandler*

tag = 'kw'

children = frozenset({'arg', 'doc', 'timeout', 'if', 'continue', 'return', 'msg', 'tag

start(*elem*, *result*)

element_handlers = {'arg': <robot.result.xmlelementhandlers.ArgumentHandler object>,

end(*elem*, *result*)

get_child_handler(*tag*)

classmethod register(*handler*)

class robot.result.xmlelementhandlers.**ForHandler**
Bases: *robot.result.xmlelementhandlers.ElementHandler*

tag = 'for'

children = frozenset({'value', 'doc', 'iter', 'msg', 'var', 'status', 'kw'})

start(*elem*, *result*)

element_handlers = {'arg': <robot.result.xmlelementhandlers.ArgumentHandler object>,

end(*elem*, *result*)

get_child_handler(*tag*)

classmethod register(*handler*)

class robot.result.xmlelementhandlers.**WhileHandler**
Bases: *robot.result.xmlelementhandlers.ElementHandler*

tag = 'while'

children = frozenset({'doc', 'msg', 'iter', 'status', 'kw'})

start(*elem*, *result*)

element_handlers = {'arg': <robot.result.xmlelementhandlers.ArgumentHandler object>,

end(*elem*, *result*)

get_child_handler(*tag*)

classmethod register(*handler*)

class robot.result.xmlelementhandlers.**IterationHandler**
Bases: *robot.result.xmlelementhandlers.ElementHandler*

tag = 'iter'

children = frozenset({'doc', 'if', 'continue', 'return', 'msg', 'break', 'var', 'statu

start(*elem*, *result*)

element_handlers = {'arg': <robot.result.xmlelementhandlers.ArgumentHandler object>,

end(*elem*, *result*)

get_child_handler(*tag*)

classmethod register(*handler*)

class robot.result.xmlelementhandlers.**IfHandler**
Bases: *robot.result.xmlelementhandlers.ElementHandler*

tag = 'if'

children = frozenset({'doc', 'msg', 'status', 'branch', 'kw'})

start(*elem*, *result*)

element_handlers = {'arg': <robot.result.xmlelementhandlers.ArgumentHandler object>,

end(*elem*, *result*)

get_child_handler(*tag*)

classmethod register(*handler*)

class robot.result.xmlelementhandlers.**BranchHandler**
Bases: *robot.result.xmlelementhandlers.ElementHandler*

tag = 'branch'

children = frozenset({'doc', 'if', 'continue', 'return', 'msg', 'pattern', 'break', 's

start(*elem*, *result*)

element_handlers = {'arg': <robot.result.xmlelementhandlers.ArgumentHandler object>,

end(*elem*, *result*)

get_child_handler(*tag*)

classmethod register(*handler*)

class robot.result.xmlelementhandlers.**TryHandler**
Bases: *robot.result.xmlelementhandlers.ElementHandler*

tag = 'try'

children = frozenset({'doc', 'msg', 'status', 'branch', 'kw'})

start(*elem*, *result*)

element_handlers = {'arg': <robot.result.xmlelementhandlers.ArgumentHandler object>,

end(*elem*, *result*)

get_child_handler(*tag*)

classmethod register(*handler*)

class robot.result.xmlelementhandlers.**PatternHandler**
Bases: *robot.result.xmlelementhandlers.ElementHandler*

tag = 'pattern'

children = frozenset()

end(*elem*, *result*)

element_handlers = {'arg': <robot.result.xmlelementhandlers.ArgumentHandler object>,

get_child_handler(*tag*)

classmethod register(*handler*)

start(*elem*, *result*)

class robot.result.xmlelementhandlers.**ReturnHandler**
Bases: *robot.result.xmlelementhandlers.ElementHandler*

tag = 'return'

children = frozenset({'value', 'msg', 'kw', 'status'})

start(*elem*, *result*)

element_handlers = {'arg': <robot.result.xmlelementhandlers.ArgumentHandler object>,

end(*elem*, *result*)

get_child_handler(*tag*)

classmethod register(*handler*)

class robot.result.xmlelementhandlers.**ContinueHandler**
Bases: *robot.result.xmlelementhandlers.ElementHandler*

tag = 'continue'

children = frozenset({'msg', 'kw', 'status'})

start(*elem*, *result*)

element_handlers = {'arg': <robot.result.xmlelementhandlers.ArgumentHandler object>,

end(*elem*, *result*)

get_child_handler(*tag*)

classmethod register(*handler*)

class robot.result.xmlelementhandlers.**BreakHandler**
Bases: *robot.result.xmlelementhandlers.ElementHandler*

tag = 'break'

children = frozenset({'msg', 'kw', 'status'})

start(*elem*, *result*)

element_handlers = {'arg': <robot.result.xmlelementhandlers.ArgumentHandler object>,

end(*elem*, *result*)

get_child_handler(*tag*)

classmethod register(*handler*)

class robot.result.xmlelementhandlers.**MessageHandler**
Bases: *robot.result.xmlelementhandlers.ElementHandler*

tag = 'msg'

end(*elem*, *result*)

children = frozenset()

element_handlers = {'arg': <robot.result.xmlelementhandlers.ArgumentHandler object>,

get_child_handler(*tag*)

classmethod register(*handler*)

start(*elem*, *result*)

class robot.result.xmlelementhandlers.**StatusHandler**(*set_status=True*)
Bases: *robot.result.xmlelementhandlers.ElementHandler*

tag = 'status'

end(*elem*, *result*)

children = frozenset()

element_handlers = {'arg': <robot.result.xmlelementhandlers.ArgumentHandler object>,

get_child_handler(*tag*)

classmethod register(*handler*)

start(*elem*, *result*)

class robot.result.xmlelementhandlers.**DocHandler**
Bases: *robot.result.xmlelementhandlers.ElementHandler*

tag = 'doc'

end(*elem*, *result*)

children = frozenset()

element_handlers = {'arg': <robot.result.xmlelementhandlers.ArgumentHandler object>,

get_child_handler(*tag*)

classmethod register(*handler*)

start(*elem*, *result*)

class robot.result.xmlelementhandlers.**MetadataHandler**
Bases: *robot.result.xmlelementhandlers.ElementHandler*

tag = 'metadata'

children = frozenset({'item'})

element_handlers = {'arg': <robot.result.xmlelementhandlers.ArgumentHandler object>,

end(*elem*, *result*)

get_child_handler(*tag*)

classmethod register(*handler*)

start(*elem*, *result*)

class robot.result.xmlelementhandlers.**MetadataItemHandler**
Bases: *robot.result.xmlelementhandlers.ElementHandler*

tag = 'item'

end(*elem*, *result*)

children = frozenset()

element_handlers = {'arg': <robot.result.xmlelementhandlers.ArgumentHandler object>,

get_child_handler(*tag*)

classmethod register(*handler*)

start(*elem*, *result*)

class robot.result.xmlelementhandlers.**MetaHandler**
Bases: *robot.result.xmlelementhandlers.ElementHandler*

tag = 'meta'

end(*elem*, *result*)

children = frozenset()

element_handlers = {'arg': <robot.result.xmlelementhandlers.ArgumentHandler object>,

get_child_handler(*tag*)

classmethod register(*handler*)

start(*elem*, *result*)

class robot.result.xmlelementhandlers.**TagsHandler**
Bases: *robot.result.xmlelementhandlers.ElementHandler*

tag = 'tags'

children = frozenset({'tag'})

element_handlers = {'arg': <robot.result.xmlelementhandlers.ArgumentHandler object>,

end(*elem*, *result*)

get_child_handler(*tag*)

classmethod register(*handler*)

start(*elem*, *result*)

class robot.result.xmlelementhandlers.**TagHandler**
Bases: *robot.result.xmlelementhandlers.ElementHandler*

tag = 'tag'

end(*elem*, *result*)

children = frozenset()

element_handlers = {'arg': <robot.result.xmlelementhandlers.ArgumentHandler object>,

get_child_handler(*tag*)

classmethod register(*handler*)

start(*elem*, *result*)

class robot.result.xmlelementhandlers.**TimeoutHandler**
Bases: *robot.result.xmlelementhandlers.ElementHandler*

tag = 'timeout'

end(*elem*, *result*)

children = frozenset()

element_handlers = {'arg': <robot.result.xmlelementhandlers.ArgumentHandler object>,

get_child_handler(*tag*)

classmethod register(*handler*)

start(*elem*, *result*)

class robot.result.xmlelementhandlers.**AssignHandler**

Bases: *robot.result.xmlelementhandlers.ElementHandler*

tag = 'assign'

children = frozenset({'var'})

element_handlers = {'arg': <robot.result.xmlelementhandlers.ArgumentHandler object>,

end(*elem*, *result*)

get_child_handler(*tag*)

classmethod register(*handler*)

start(*elem*, *result*)

class robot.result.xmlelementhandlers.**VarHandler**

Bases: *robot.result.xmlelementhandlers.ElementHandler*

tag = 'var'

end(*elem*, *result*)

children = frozenset()

element_handlers = {'arg': <robot.result.xmlelementhandlers.ArgumentHandler object>,

get_child_handler(*tag*)

classmethod register(*handler*)

start(*elem*, *result*)

class robot.result.xmlelementhandlers.**ArgumentsHandler**

Bases: *robot.result.xmlelementhandlers.ElementHandler*

tag = 'arguments'

children = frozenset({'arg'})

element_handlers = {'arg': <robot.result.xmlelementhandlers.ArgumentHandler object>,

end(*elem*, *result*)

get_child_handler(*tag*)

classmethod register(*handler*)

start(*elem*, *result*)

class robot.result.xmlelementhandlers.**ArgumentHandler**

Bases: *robot.result.xmlelementhandlers.ElementHandler*

tag = 'arg'

end(*elem*, *result*)

children = frozenset()

element_handlers = {'arg': <robot.result.xmlelementhandlers.ArgumentHandler object>,

get_child_handler(*tag*)

classmethod register(*handler*)

start(*elem*, *result*)

class robot.result.xmlelementhandlers.**ValueHandler**

Bases: *robot.result.xmlelementhandlers.ElementHandler*

tag = 'value'

end(*elem*, *result*)

children = frozenset()

element_handlers = {'arg': <robot.result.xmlelementhandlers.ArgumentHandler object>,

get_child_handler(*tag*)

classmethod register(*handler*)

start(*elem*, *result*)

class robot.result.xmlelementhandlers.**ErrorsHandler**

Bases: *robot.result.xmlelementhandlers.ElementHandler*

tag = 'errors'

start(*elem*, *result*)

get_child_handler(*tag*)

children = frozenset()

element_handlers = {'arg': <robot.result.xmlelementhandlers.ArgumentHandler object>,

end(*elem*, *result*)

classmethod register(*handler*)

class robot.result.xmlelementhandlers.**ErrorMessageHandler**

Bases: *robot.result.xmlelementhandlers.ElementHandler*

end(*elem*, *result*)

children = frozenset()

element_handlers = {'arg': <robot.result.xmlelementhandlers.ArgumentHandler object>,

get_child_handler(*tag*)

classmethod register(*handler*)

start(*elem*, *result*)

tag = None

class robot.result.xmlelementhandlers.**StatisticsHandler**

Bases: *robot.result.xmlelementhandlers.ElementHandler*

tag = 'statistics'

get_child_handler(*tag*)

children = frozenset()

element_handlers = {'arg': <robot.result.xmlelementhandlers.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*, *languages=None*)

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'*, *error_reporter=None*)
Bases: robot.running.arguments.argumentparser._ArgumentParser

parse(*handler*, *name=None*)

class robot.running.arguments.argumentparser.**DynamicArgumentParser**(*type='Keyword'*, *error_reporter=None*)
Bases: robot.running.arguments.argumentparser._ArgumentSpecParser

parse(*argspec*, *name=None*)

class robot.running.arguments.argumentparser.**UserKeywordArgumentParser**(*type='Keyword'*, *error_reporter=None*)
Bases: robot.running.arguments.argumentparser._ArgumentSpecParser

parse(*argspec*, *name=None*)

robot.running.arguments.argumentresolver module

class robot.running.arguments.argumentresolver.**ArgumentResolver**(*argspec*, *resolve_named=True*, *resolve_variables_until=None*, *dict_to_kwargs=False*)
Bases: object

resolve(*arguments*, *variables=None*)

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

resolve(*arguments*, *variables=None*)

class robot.running.arguments.argumentresolver.**NullNamedArgumentResolver**
Bases: object

resolve(*arguments*, *variables=None*)

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

handle(*positional*, *named*)

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

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

robot.running.arguments.argumentspec module

class robot.running.arguments.argumentspec.**ArgumentSpec**(*name=None*, *type='Keyword'*, *positional_only=None*, *positional_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*, *resolve_variables_until=None*, *dict_to_kwargs=False*, *languages=None*)

convert(*positional*, *named*, *converters=None*, *dry_run=False*, *languages=None*)

map(*positional*, *named*, *replace_defaults=True*)

class robot.running.arguments.argumentspec.**ArgInfo**(*kind*, *name=''*, *types=<object object>*, *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=None*, *args=()*, *custom_patterns=None*)
Bases: object

classmethod from_name(*name*)

match(*name*)

map(*values*)

validate(*values*)

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*, *languages=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*, *languages=None*)

classmethod handles(*type_*)

convert(*name*, *value*, *explicit_type=True*, *strict=True*, *kind='Argument'*)

no_conversion_needed(*value*)

class robot.running.arguments.typeconverters.**EnumConverter**(*used_type*, *custom_converters=None*, *languages=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*, *kind='Argument'*)

classmethod converter_for(*type_*, *custom_converters=None*, *languages=None*)

doc = None

classmethod handles(*type_*)

no_conversion_needed(*value*)

classmethod register(*converter*)

class robot.running.arguments.typeconverters.**StringConverter**(*used_type*, *custom_converters=None*, *languages=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*, *kind='Argument'*)

classmethod converter_for(*type_*, *custom_converters=None*, *languages=None*)

doc = None

classmethod handles(*type_*)

no_conversion_needed(*value*)

classmethod register(*converter*)

class robot.running.arguments.typeconverters.**BooleanConverter**(*used_type*, *custom_converters=None*, *languages=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*, *kind='Argument'*)

classmethod converter_for(*type_*, *custom_converters=None*, *languages=None*)

doc = None

classmethod handles(*type_*)

no_conversion_needed(*value*)

classmethod register(*converter*)

class robot.running.arguments.typeconverters.**IntegerConverter**(*used_type*, *custom_converters=None*, *languages=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*, *kind='Argument'*)

classmethod converter_for(*type_*, *custom_converters=None*, *languages=None*)

doc = None

classmethod handles(*type_*)

no_conversion_needed(*value*)

classmethod register(*converter*)

class robot.running.arguments.typeconverters.**FloatConverter**(*used_type*, *custom_converters=None*, *languages=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*, *kind='Argument'*)

classmethod converter_for(*type_*, *custom_converters=None*, *languages=None*)

doc = None

classmethod handles(*type_*)

no_conversion_needed(*value*)

classmethod register(*converter*)

class robot.running.arguments.typeconverters.**DecimalConverter**(*used_type*, *custom_converters=None*, *languages=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*, *kind='Argument'*)

classmethod converter_for(*type_*, *custom_converters=None*, *languages=None*)

doc = None

classmethod handles(*type_*)

no_conversion_needed(*value*)

classmethod register(*converter*)

class robot.running.arguments.typeconverters.**BytesConverter**(*used_type*, *custom_converters=None*, *languages=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*, *kind='Argument'*)

classmethod converter_for(*type_*, *custom_converters=None*, *languages=None*)

doc = None

classmethod handles(*type_*)

no_conversion_needed(*value*)

classmethod register(*converter*)

class robot.running.arguments.typeconverters.**ByteArrayConverter**(*used_type*, *custom_converters=None*, *languages=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*, *kind='Argument'*)

classmethod converter_for(*type_*, *custom_converters=None*, *languages=None*)

doc = None

classmethod handles(*type_*)

no_conversion_needed(*value*)

classmethod register(*converter*)

class robot.running.arguments.typeconverters.**DateTimeConverter**(*used_type*, *custom_converters=None*, *languages=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*, *kind='Argument'*)

classmethod converter_for(*type_*, *custom_converters=None*, *languages=None*)

doc = None

classmethod handles(*type_*)

no_conversion_needed(*value*)

classmethod register(*converter*)

class robot.running.arguments.typeconverters.**DateConverter**(*used_type*, *custom_converters=None*, *languages=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*, *kind='Argument'*)

classmethod converter_for(*type_*, *custom_converters=None*, *languages=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*, *custom_converters=None*, *languages=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*, *kind='Argument'*)

classmethod converter_for(*type_*, *custom_converters=None*, *languages=None*)

doc = None

classmethod handles(*type_*)

no_conversion_needed(*value*)

classmethod register(*converter*)

class robot.running.arguments.typeconverters.**PathConverter**(*used_type*, *custom_converters=None*, *languages=None*)

Bases: *robot.running.arguments.typeconverters.TypeConverter*

type
alias of pathlib.Path

abc
alias of os.PathLike

type_name = 'Path'

value_types = (<class 'str'>, <class 'pathlib.PurePath'>)

aliases = ()

convert(*name*, *value*, *explicit_type=True*, *strict=True*, *kind='Argument'*)

classmethod converter_for(*type_*, *custom_converters=None*, *languages=None*)

doc = None

classmethod handles(*type_*)

no_conversion_needed(*value*)

classmethod register(*converter*)

class robot.running.arguments.typeconverters.**NoneConverter**(*used_type*, *custom_converters=None*, *languages=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*, *kind='Argument'*)

classmethod converter_for(*type_*, *custom_converters=None*, *languages=None*)

doc = None

no_conversion_needed(*value*)

classmethod register(*converter*)

value_types = (<class 'str'>,)

class robot.running.arguments.typeconverters.**ListConverter**(*used_type*, *custom_converters=None*, *languages=None*)

Bases: *robot.running.arguments.typeconverters.TypeConverter*

type
alias of builtins.list

abc
alias of collections.abc.Sequence

value_types = (<class 'str'>, <class 'collections.abc.Sequence'>)

type_name = 'list'

classmethod handles(*type_*)

no_conversion_needed(*value*)

aliases = ()

convert(*name*, *value*, *explicit_type=True*, *strict=True*, *kind='Argument'*)

classmethod converter_for(*type_*, *custom_converters=None*, *languages=None*)

doc = None

classmethod register(*converter*)

class robot.running.arguments.typeconverters.**TupleConverter**(*used_type*, *custom_converters=None*, *languages=None*)

Bases: *robot.running.arguments.typeconverters.TypeConverter*

type
alias of builtins.tuple

value_types = (<class 'str'>, <class 'collections.abc.Sequence'>)

type_name = 'tuple'

abc = None

aliases = ()

convert(*name*, *value*, *explicit_type=True*, *strict=True*, *kind='Argument'*)

classmethod converter_for(*type_*, *custom_converters=None*, *languages=None*)

doc = None

classmethod handles(*type_*)

no_conversion_needed(*value*)

classmethod register(*converter*)

class robot.running.arguments.typeconverters.**TypedDictConverter**(*used_type*, *custom_converters*, *languages=None*)

Bases: *robot.running.arguments.typeconverters.TypeConverter*

type = 'TypedDict'

value_types = (<class 'str'>, <class 'collections.abc.Mapping'>)

type_name = None

classmethod handles(*type_*)

no_conversion_needed(*value*)

abc = None

aliases = ()

convert(*name*, *value*, *explicit_type=True*, *strict=True*, *kind='Argument'*)

classmethod converter_for(*type_*, *custom_converters=None*, *languages=None*)

doc = None

classmethod register(*converter*)

class robot.running.arguments.typeconverters.**DictionaryConverter**(*used_type*, *custom_converters=None*, *languages=None*)

Bases: *robot.running.arguments.typeconverters.TypeConverter*

type
alias of builtins.dict

abc
alias of collections.abc.Mapping

aliases = ('dict', 'map')

value_types = (<class 'str'>, <class 'collections.abc.Mapping'>)

type_name = 'dictionary'

convert(*name*, *value*, *explicit_type=True*, *strict=True*, *kind='Argument'*)

classmethod converter_for(*type_*, *custom_converters=None*, *languages=None*)

doc = None

classmethod handles(*type_*)

no_conversion_needed(*value*)

classmethod register(*converter*)

class robot.running.arguments.typeconverters.**SetConverter**(*used_type*, *custom_converters=None*, *languages=None*)

Bases: *robot.running.arguments.typeconverters.TypeConverter*

type
alias of builtins.set

abc
alias of collections.abc.Set

value_types = (<class 'str'>, <class 'collections.abc.Container'>)

type_name = 'set'

aliases = ()

convert(*name*, *value*, *explicit_type=True*, *strict=True*, *kind='Argument'*)

classmethod converter_for(*type_*, *custom_converters=None*, *languages=None*)

doc = None

classmethod handles(*type_*)

no_conversion_needed(*value*)

classmethod register(*converter*)

class robot.running.arguments.typeconverters.**FrozenSetConverter**(*used_type*, *custom_converters=None*, *languages=None*)

Bases: *robot.running.arguments.typeconverters.SetConverter*

type
alias of builtins.frozenset

type_name = 'frozenset'

abc
alias of collections.abc.Set

aliases = ()

convert(*name*, *value*, *explicit_type=True*, *strict=True*, *kind='Argument'*)

classmethod converter_for(*type_*, *custom_converters=None*, *languages=None*)

doc = None

classmethod handles(*type_*)

no_conversion_needed(*value*)

classmethod register(*converter*)

value_types = (<class 'str'>, <class 'collections.abc.Container'>)

class robot.running.arguments.typeconverters.**CombinedConverter**(*union*, *custom_converters*, *languages=None*)

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*, *kind='Argument'*)

classmethod converter_for(*type_*, *custom_converters=None*, *languages=None*)

doc = None

classmethod register(*converter*)

value_types = (<class 'str'>,)

class robot.running.arguments.typeconverters.**CustomConverter**(*used_type*, *converter_info*, *languages=None*)

Bases: *robot.running.arguments.typeconverters.TypeConverter*

abc = None

aliases = ()

convert(*name*, *value*, *explicit_type=True*, *strict=True*, *kind='Argument'*)

classmethod converter_for(*type_*, *custom_converters=None*, *languages=None*)

classmethod handles(*type_*)

no_conversion_needed(*value*)

classmethod register(*converter*)

type = None

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.

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*, *lang=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*.
- **lang** – Additional languages to be supported during parsing. Can be a string matching any of the supported language codes or names, an initialized *Language* subsclass, a list containing such strings or instances, or a *Languages* instance.
- **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*, *lang=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**(*lang=None*, *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**(*lang=None*, *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**(*lang=None*, *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.settings module

class robot.running.builder.settings.**Defaults**(*parent=None*)
Bases: object

setup

teardown

force_tags

timeout

class robot.running.builder.settings.**TestSettings**(*defaults*)
Bases: object

setup

teardown

timeout

template

tags

robot.running.builder.transformers module

class robot.running.builder.transformers.**SettingsBuilder**(*suite*, *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_KeywordTags(*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*, *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_KeywordTags(*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*, *defaults*)
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.builder.transformers.**deprecate_tags_starting_with_hyphen**(*node*, *source*)

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*, *templated=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

classmethod create(*limit*, *variables*)

limit_exceeded()

class robot.running.bodyrunner.**DurationLimit**(*max_time*)
Bases: *robot.running.bodyrunner.WhileLimit*

classmethod create(*limit*, *variables*)

limit_exceeded()

class robot.running.bodyrunner.**IterationCountLimit**(*max_iterations*)
Bases: *robot.running.bodyrunner.WhileLimit*

classmethod create(*limit*, *variables*)

limit_exceeded()

class robot.running.bodyrunner.**NoLimit**
Bases: *robot.running.bodyrunner.WhileLimit*

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**(*embedded*, *orig_handler*)
Bases: object

supports_embedded_args = True

library

matches(*name*)

create_runner(*name*, *languages=None*)

resolve_arguments(*args*, *variables=None*, *languages=None*)

robot.running.handlerstore module

class robot.running.handlerstore.**HandlerStore**
Bases: object

add(*handler*, *embedded=False*)

get_handlers(*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*, *lang=None*)

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*, *languages=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*, *execute_in_dry_run=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 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.

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 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
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*, *variable=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 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 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 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 = ('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 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 = '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 about 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 about 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*, *timeout=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*, *languages*)
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*, *recommend_on_failure=True*)

class robot.running.namespace.**KeywordStore**(*resource*, *languages*)
Bases: object

get_library(*name_or_instance*)

get_runner(*name*, *recommend=True*)

class robot.running.namespace.**KeywordRecommendationFinder**(*user_keywords*, *libraries*, *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*, *suppress=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.

supports_embedded_arguments = False

longname

doc

shortdoc

create_runner(*name*, *languages=None*)

run(*kw*, *context*, *run=True*)

dry_run(*kw*, *context*, *run=True*)

robot.running.userkeyword module

class robot.running.userkeyword.**UserLibrary**(*resource*, *resource_file=True*)
Bases: `object`

handlers_for(*name*)

class robot.running.userkeyword.**UserKeywordHandler**(*keyword*, *libname*)
Bases: object

supports_embedded_args = False

longname

shortdoc

private

create_runner(*name*, *languages=None*)

class robot.running.userkeyword.**EmbeddedArgumentsHandler**(*keyword*, *libname*, *embedded*)
Bases: *robot.running.userkeyword.UserKeywordHandler*

supports_embedded_args = True

matches(*name*)

create_runner(*name*, *languages=None*)

longname

private

shortdoc

robot.running.userkeywordrunner module

class robot.running.userkeywordrunner.**UserKeywordRunner**(*handler*, *name=None*)
Bases: object

longname

libname

tags

source

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*)

source

tags

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*, *logger=None*, ***auto_options*)

Bases: object

main(*arguments*, ***options*)

validate(*options*, *arguments*)

execute_cli(*cli_arguments*, *exit=True*)

console(*msg*)

parse_arguments(*cli_args*)
Public interface for parsing command line arguments.

Parameters **cli_args** – Command line arguments as a list

Returns options (dict), arguments (list)

Raises *Information* when –help or –version used

Raises *DataError* when parsing fails

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

class robot.utils.application.**DefaultLogger**

Bases: object

info(*message*)

error(*message*)

close()

robot.utils.argumentparser module

robot.utils.argumentparser.**cmdline2list**(*args*, *escaping=False*)

class robot.utils.argumentparser.**ArgumentParser**(*usage*, *name=None*, *version=None*, *arg_limits=None*, *validator=None*, *env_options=None*, *auto_help=True*, *auto_version=True*, *auto_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 AssertionErrors occur outside.

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

unittest license:

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

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

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

Examples:

```
import unittest
from robot.utils.asserts import assert_equal

class MyTests(unittest.TestCase):

    def test_old_style(self):
        self.assertEqual(1, 2, 'my msg')

    def test_new_style(self):
        assert_equal(1, 2, 'my msg')
```

Example output:

```
FF
======================================================================
FAIL: test_old_style (example.MyTests)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "example.py", line 7, in test_old_style
    self.assertEqual(1, 2, 'my msg')
AssertionError: my msg

======================================================================
FAIL: test_new_style (example.MyTests)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "example.py", line 10, in test_new_style
    assert_equal(1, 2, 'my msg')
  File "/path/to/robot/utils/asserts.py", line 181, in assert_equal
    _report_inequality_failure(first, second, msg, values, '!=')
  File "/path/to/robot/utils/asserts.py", line 229, in _report_inequality_failure
    raise AssertionError(msg)
AssertionError: my msg: 1 != 2

----------------------------------------------------------------------
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*, ***kwds*)
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*, *exclude_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.

Use *import_module()* if only a module needs to be imported.

import_module(*name_or_path*)
Imports Python module based on the given name or path.

Parameters **name_or_path** – Name or path of the module to import.

The 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 modules by path.

Use *import_class_or_module()* if it is desired to get a class from the imported module automatically.

New in Robot Framework 6.0.

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*, *get_class=True*)

class robot.utils.importer.**NonDottedImporter**(*logger*)
Bases: robot.utils.importer._Importer

handles(*name*)

import_(*name*, *get_class=True*)

class robot.utils.importer.**DottedImporter**(*logger*)
Bases: robot.utils.importer._Importer

handles(*name*)

import_(*name*, *get_class=True*)

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*, *regexp=False*)
Bases: object

match(*string*)

match_any(*strings*)

class robot.utils.match.**MultiMatcher**(*patterns=None*, *ignore=()*, *caseless=True*, *spaceless=True*, *match_if_no_patterns=False*, *regexp=False*)
Bases: object

match(*string*)

match_any(*strings*)

robot.utils.misc module

robot.utils.misc.**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: str*, *rpa: bool*)
Replace 'test' with 'task' in the given *text* depending on *rpa*.

If given text is *test*, *test* or *task* is returned directly. Otherwise, pattern *{test}* is searched from the text and occurrences replaced with *test* or *task*.

In both cases matching the word *test* is case-insensitive and the returned *test* or *task* has exactly same case as the original.

robot.utils.misc.**isatty**(*stream*)

robot.utils.misc.**parse_re_flags**(*flags=None*)

class robot.utils.misc.**classproperty**(*fget*, *fset=None*, *fdel=None*, *doc=None*)
Bases: property

Property that works with classes in addition to instances.

Only supports getters. Setters and deleters cannot work with classes due to how the descriptor protocol works, and they are thus explicitly disabled. Metaclasses must be used if they are needed.

setter(*fset*)
Descriptor to change the setter on a property.

deleter(*fset*)
Descriptor to change the deleter on a property.

fdel

fget

fset

getter()
Descriptor to change the getter on a property.

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*, *modules=None*, *namespace=None*)

class robot.variables.evaluation.**EvaluationNamespace**(*variable_store*, *namespace*)
Bases: collections.abc.MutableMapping

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.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*)

robot.variables.search.**unescape_variable_syntax**(*item*)

class robot.variables.search.**VariableIterator**(*string*, *identifiers='$@&%'*, *ignore_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*, *error_reporter=None*)
Bases: *robot.variables.resolvable.Resolvable*

resolve(*variables*)

report_error(*error*)

class robot.variables.tablesetter.**ScalarVariableTableValue**(*values*, *error_reporter=None*)
Bases: *robot.variables.tablesetter.VariableTableValueBase*

report_error(*error*)

resolve(*variables*)

class robot.variables.tablesetter.**ListVariableTableValue**(*values*, *error_reporter=None*)
Bases: *robot.variables.tablesetter.VariableTableValueBase*

report_error(*error*)

resolve(*variables*)

class robot.variables.tablesetter.**DictVariableTableValue**(*values*, *error_reporter=None*)
Bases: *robot.variables.tablesetter.VariableTableValueBase*

report_error(*error*)

resolve(*variables*)

robot.variables.variables module

class robot.variables.variables.**Variables**
Bases: object

Represents a set of variables.

Contains methods for replacing variables from list, scalars, and strings. On top of ${scalar}, @{list} and &{dict} variables, these methods handle also %{environment} variables.

resolve_delayed()

replace_list(*items*, *replace_until=None*, *ignore_errors=False*)

replace_scalar(*item*, *ignore_errors=False*)

replace_string(*item*, *custom_unescaper=None*, *ignore_errors=False*)

set_from_file(*path_or_variables*, *args=None*, *overwrite=False*)

set_from_variable_table(*variables*, *overwrite=False*)

clear()

copy()

update(*variables*)

as_dict(*decoration=True*)

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=''*, *syntax=False*)
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*, *teardown_errors=None*)
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
```

This module also exposes the following public API:

- *libdoc_cli()* function for simple command line tools.

- *libdoc()* function as a high level programmatic API.
- *LibraryDocumentation()* as the API to generate *LibraryDoc* instances.

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*, *theme=None*, *pythonpath=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 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')
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

Python Module Index

r

robot, 7
robot.api, 5
robot.api.deco, 11
robot.api.exceptions, 12
robot.api.logger, 14
robot.api.parsing, 15
robot.conf, 23
robot.conf.gatherfailed, 23
robot.conf.languages, 32
robot.conf.settings, 65
robot.errors, 613
robot.htmldata, 67
robot.htmldata.htmlfilewriter, 67
robot.htmldata.jsonwriter, 68
robot.htmldata.template, 68
robot.libdoc, 618
robot.libdocpkg, 69
robot.libdocpkg.builder, 69
robot.libdocpkg.consoleviewer, 69
robot.libdocpkg.datatypes, 70
robot.libdocpkg.htmlutils, 70
robot.libdocpkg.htmlwriter, 71
robot.libdocpkg.jsonbuilder, 71
robot.libdocpkg.jsonwriter, 71
robot.libdocpkg.model, 71
robot.libdocpkg.output, 72
robot.libdocpkg.robotbuilder, 72
robot.libdocpkg.standardtypes, 72
robot.libdocpkg.writer, 72
robot.libdocpkg.xmlbuilder, 73
robot.libdocpkg.xmlwriter, 73
robot.libraries, 73
robot.libraries.BuiltIn, 73
robot.libraries.Collections, 100
robot.libraries.DateTime, 106
robot.libraries.Dialogs, 111
robot.libraries.dialogs_py, 157
robot.libraries.Easter, 112
robot.libraries.OperatingSystem, 112
robot.libraries.Process, 122
robot.libraries.Remote, 128
robot.libraries.Reserved, 130
robot.libraries.Screenshot, 130
robot.libraries.String, 131
robot.libraries.Telnet, 137
robot.libraries.XML, 147
robot.model, 227
robot.model.body, 228
robot.model.configurer, 234
robot.model.control, 238
robot.model.filter, 250
robot.model.fixture, 258
robot.model.itemlist, 259
robot.model.keyword, 259
robot.model.message, 262
robot.model.metadata, 264
robot.model.modelobject, 265
robot.model.modifier, 265
robot.model.namepatterns, 270
robot.model.statistics, 270
robot.model.stats, 274
robot.model.suitestatistics, 276
robot.model.tags, 277
robot.model.tagsetter, 277
robot.model.tagstatistics, 282
robot.model.testcase, 282
robot.model.testsuite, 285
robot.model.totalstatistics, 288
robot.model.visitor, 293
robot.output, 298
robot.output.console, 298
robot.output.console.dotted, 298
robot.output.console.highlighting, 303
robot.output.console.quiet, 304
robot.output.console.verbose, 304
robot.output.debugfile, 305
robot.output.filelogger, 305
robot.output.librarylogger, 305

robot.output.listenerarguments, 306
robot.output.listenermethods, 307
robot.output.listeners, 307
robot.output.logger, 308
robot.output.loggerhelper, 309
robot.output.output, 311
robot.output.pyloggingconf, 312
robot.output.stdoutlogsplitter, 313
robot.output.xmllogger, 313
robot.parsing, 318
robot.parsing.lexer, 318
robot.parsing.lexer.blocklexers, 318
robot.parsing.lexer.context, 323
robot.parsing.lexer.lexer, 325
robot.parsing.lexer.settings, 326
robot.parsing.lexer.statementlexers, 327
robot.parsing.lexer.tokenizer, 333
robot.parsing.lexer.tokens, 333
robot.parsing.model, 340
robot.parsing.model.blocks, 340
robot.parsing.model.statements, 345
robot.parsing.model.visitor, 388
robot.parsing.parser, 389
robot.parsing.parser.blockparsers, 389
robot.parsing.parser.fileparser, 390
robot.parsing.parser.parser, 391
robot.parsing.suitestructure, 392
robot.pythonpathsetter, 620
robot.rebot, 620
robot.reporting, 393
robot.reporting.expandkeywordmatcher, 393
robot.reporting.jsbuildingcontext, 393
robot.reporting.jsexecutionresult, 394
robot.reporting.jsmodelbuilders, 394
robot.reporting.jswriter, 395
robot.reporting.logreportwriters, 395
robot.reporting.outputwriter, 395
robot.reporting.resultwriter, 400
robot.reporting.stringcache, 401
robot.reporting.xunitwriter, 402
robot.result, 407
robot.result.configurer, 408
robot.result.executionerrors, 412
robot.result.executionresult, 413
robot.result.flattenkeywordmatcher, 415
robot.result.keywordremover, 415
robot.result.merger, 449
robot.result.messagefilter, 453
robot.result.model, 457
robot.result.modeldeprecation, 499
robot.result.resultbuilder, 500
robot.result.suiteteardownfailed, 505
robot.result.visitor, 513
robot.result.xmlelementhandlers, 519
robot.run, 621
robot.running, 527
robot.running.arguments, 528
robot.running.arguments.argumentconverter, 528
robot.running.arguments.argumentmapper, 528
robot.running.arguments.argumentparser, 529
robot.running.arguments.argumentresolver, 529
robot.running.arguments.argumentspec, 530
robot.running.arguments.argumentvalidator, 531
robot.running.arguments.customconverters, 531
robot.running.arguments.embedded, 531
robot.running.arguments.typeconverters, 531
robot.running.arguments.typevalidator, 541
robot.running.bodyrunner, 549
robot.running.builder, 541
robot.running.builder.builders, 541
robot.running.builder.parsers, 542
robot.running.builder.settings, 543
robot.running.builder.transformers, 544
robot.running.context, 550
robot.running.dynamicmethods, 550
robot.running.handlers, 551
robot.running.handlerstore, 551
robot.running.importer, 552
robot.running.librarykeywordrunner, 552
robot.running.libraryscopes, 553
robot.running.model, 553
robot.running.modelcombiner, 577
robot.running.namespace, 577
robot.running.outputcapture, 578
robot.running.randomizer, 578
robot.running.runkwregister, 583
robot.running.signalhandler, 583
robot.running.status, 583
robot.running.statusreporter, 584
robot.running.suiterunner, 585
robot.running.testlibraries, 589
robot.running.timeouts, 548
robot.running.timeouts.posix, 549
robot.running.timeouts.windows, 549
robot.running.usererrorhandler, 589
robot.running.userkeyword, 589
robot.running.userkeywordrunner, 590
robot.testdoc, 623

robot.utils, 591
robot.utils.application, 591
robot.utils.argumentparser, 592
robot.utils.asserts, 593
robot.utils.charwidth, 595
robot.utils.compress, 595
robot.utils.connectioncache, 595
robot.utils.dotdict, 596
robot.utils.encoding, 597
robot.utils.encodingsniffer, 597
robot.utils.error, 598
robot.utils.escaping, 598
robot.utils.etreewrapper, 598
robot.utils.filereader, 598
robot.utils.frange, 599
robot.utils.htmlformatters, 599
robot.utils.importer, 600
robot.utils.markuputils, 602
robot.utils.markupwriters, 602
robot.utils.match, 603
robot.utils.misc, 603
robot.utils.normalizing, 604
robot.utils.platform, 605
robot.utils.recommendations, 605
robot.variables, 606
robot.variables.assigner, 606
robot.variables.evaluation, 607
robot.variables.filesetter, 607
robot.variables.finders, 608
robot.variables.notfound, 608
robot.variables.replacer, 609
robot.variables.resolvable, 609
robot.variables.scopes, 609
robot.variables.search, 611
robot.variables.store, 611
robot.variables.tablesetter, 612
robot.variables.variables, 612
robot.version, 625

Index

A

abc (*robot.running.arguments.typeconverters.BooleanConverter attribute*), 533
abc (*robot.running.arguments.typeconverters.ByteArrayConverter attribute*), 535
abc (*robot.running.arguments.typeconverters.BytesConverter attribute*), 534
abc (*robot.running.arguments.typeconverters.CombinedConverter attribute*), 540
abc (*robot.running.arguments.typeconverters.CustomConverter attribute*), 540
abc (*robot.running.arguments.typeconverters.DateConverter attribute*), 536
abc (*robot.running.arguments.typeconverters.DateTimeConverter attribute*), 535
abc (*robot.running.arguments.typeconverters.DecimalConverter attribute*), 534
abc (*robot.running.arguments.typeconverters.DictionaryConverter attribute*), 539
abc (*robot.running.arguments.typeconverters.EnumConverter attribute*), 532
abc (*robot.running.arguments.typeconverters.FloatConverter attribute*), 534
abc (*robot.running.arguments.typeconverters.FrozenSetConverter attribute*), 540
abc (*robot.running.arguments.typeconverters.IntegerConverter attribute*), 533
abc (*robot.running.arguments.typeconverters.ListConverter attribute*), 537
abc (*robot.running.arguments.typeconverters.NoneConverter attribute*), 537
abc (*robot.running.arguments.typeconverters.PathConverter attribute*), 537
abc (*robot.running.arguments.typeconverters.SetConverter attribute*), 539
abc (*robot.running.arguments.typeconverters.StringConverter attribute*), 532
abc (*robot.running.arguments.typeconverters.TimeDeltaConverter attribute*), 536
abc (*robot.running.arguments.typeconverters.TupleConverter attribute*), 538
abc (*robot.running.arguments.typeconverters.TypeConverter attribute*), 531
abc (*robot.running.arguments.typeconverters.TypedDictConverter attribute*), 538
AbstractLogger (*class in robot.output.loggerhelper*), 309
AbstractLoggerProxy (*class in robot.output.loggerhelper*), 311
accept_gzip_encoding (*robot.libraries.Remote.TimeoutHTTPSTransport attribute*), 129
accept_gzip_encoding (*robot.libraries.Remote.TimeoutHTTPTransport attribute*), 129
accepts_more() (*robot.parsing.lexer.blocklexers.BlockLexer method*), 318
accepts_more() (*robot.parsing.lexer.blocklexers.CommentSectionLexer method*), 320
accepts_more() (*robot.parsing.lexer.blocklexers.ErrorSectionLexer method*), 321
accepts_more() (*robot.parsing.lexer.blocklexers.FileLexer method*), 319
accepts_more() (*robot.parsing.lexer.blocklexers.ForLexer method*), 322
accepts_more() (*robot.parsing.lexer.blocklexers.IfLexer method*), 322
accepts_more() (*robot.parsing.lexer.blocklexers.ImplicitCommentSect method*), 321
accepts_more() (*robot.parsing.lexer.blocklexers.InlineIfLexer method*), 323
accepts_more() (*robot.parsing.lexer.blocklexers.KeywordLexer method*), 322
accepts_more() (*robot.parsing.lexer.blocklexers.KeywordSectionLexer method*), 320
accepts_more() (*robot.parsing.lexer.blocklexers.NestedBlockLexer method*), 322
accepts_more() (*robot.parsing.lexer.blocklexers.SectionLexer method*), 319

accepts_more() (*robot.parsing.lexer.blocklexers.SettingSectionLexer method*), 319
accepts_more() (*robot.parsing.lexer.blocklexers.TaskSectionLexer method*), 320
accepts_more() (*robot.parsing.lexer.blocklexers.TestCaseLexer method*), 321
accepts_more() (*robot.parsing.lexer.blocklexers.TestCaseSectionLexer method*), 320
accepts_more() (*robot.parsing.lexer.blocklexers.TestOrKeywordLexer method*), 321
accepts_more() (*robot.parsing.lexer.blocklexers.TryLexer method*), 323
accepts_more() (*robot.parsing.lexer.blocklexers.VariableSectionLexer method*), 319
accepts_more() (*robot.parsing.lexer.blocklexers.WhileLexer method*), 322
accepts_more() (*robot.parsing.lexer.statementlexers.BreakLexer method*), 333
accepts_more() (*robot.parsing.lexer.statementlexers.CommentLexer method*), 330
accepts_more() (*robot.parsing.lexer.statementlexers.CommentSectionHeaderLexer method*), 329
accepts_more() (*robot.parsing.lexer.statementlexers.ContinueLexer method*), 333
accepts_more() (*robot.parsing.lexer.statementlexers.ElseHeaderLexer method*), 331
accepts_more() (*robot.parsing.lexer.statementlexers.ElseIfHeaderLexer method*), 331
accepts_more() (*robot.parsing.lexer.statementlexers.EndLexer method*), 332
accepts_more() (*robot.parsing.lexer.statementlexers.ErrorSectionHeaderLexer method*), 329
accepts_more() (*robot.parsing.lexer.statementlexers.ExceptHeaderLexer method*), 332
accepts_more() (*robot.parsing.lexer.statementlexers.FinallyHeaderLexer method*), 332
accepts_more() (*robot.parsing.lexer.statementlexers.ForHeaderLexer method*), 331
accepts_more() (*robot.parsing.lexer.statementlexers.IfHeaderLexer method*), 331
accepts_more() (*robot.parsing.lexer.statementlexers.ImplicitCommentLexer method*), 330
accepts_more() (*robot.parsing.lexer.statementlexers.InlineIfHeaderLexer method*), 331
accepts_more() (*robot.parsing.lexer.statementlexers.KeywordCallLexer method*), 330
accepts_more() (*robot.parsing.lexer.statementlexers.KeywordSectionHeaderLexer method*), 329
accepts_more() (*robot.parsing.lexer.statementlexers.Lexer method*), 327
accepts_more() (*robot.parsing.lexer.statementlexers.ReturnLexer method*), 333
accepts_more() (*robot.parsing.lexer.statementlexers.SectionHeaderLexer method*), 328
accepts_more() (*robot.parsing.lexer.statementlexers.SettingLexer method*), 330
accepts_more() (*robot.parsing.lexer.statementlexers.SettingSectionHeaderLexer method*), 328
accepts_more() (*robot.parsing.lexer.statementlexers.SingleType method*), 328
accepts_more() (*robot.parsing.lexer.statementlexers.StatementLexer method*), 327
accepts_more() (*robot.parsing.lexer.statementlexers.TaskSectionHeaderLexer method*), 329
accepts_more() (*robot.parsing.lexer.statementlexers.TestCaseSectionHeaderLexer method*), 329
accepts_more() (*robot.parsing.lexer.statementlexers.TestOrKeywordSettingLexer method*), 330
accepts_more() (*robot.parsing.lexer.statementlexers.TryHeaderLexer method*), 332
accepts_more() (*robot.parsing.lexer.statementlexers.TypeAndArguments method*), 328
accepts_more() (*robot.parsing.lexer.statementlexers.VariableLexer method*), 330
accepts_more() (*robot.parsing.lexer.statementlexers.VariableSectionHeaderLexer method*), 328
accepts_more() (*robot.parsing.lexer.statementlexers.WhileHeaderLexer method*), 332
add_handler() (*robot.output.pyloggingconf.RobotHandler method*), 312
add_keyword_timeout() (*robot.running.timeouts.KeywordTimeout attribute*), 548
add_timeout() (*robot.running.timeouts.TestTimeout attribute*), 548
add() (*robot.model.tagsetters.Tags method*), 277
add() (*robot.reporting.stringcache.StringCache method*), 402
add() (*robot.result.executionerrors.ExecutionErrors method*), 413
add() (*robot.running.handlerstore.HandlerStore method*), 551
add() (*robot.running.importer.ImportCache method*), 552
add() (*robot.utils.htmlformatters.HeaderFormatter method*), 599
add() (*robot.utils.htmlformatters.ListFormatter method*), 600
add() (*robot.utils.htmlformatters.ParagraphFormatter method*), 600
add() (*robot.utils.htmlformatters.PreformattedFormatter method*), 601
add() (*robot.utils.htmlformatters.RulerFormatter method*), 599
add() (*robot.utils.htmlformatters.TableFormatter method*), 600
add() (*robot.variables.store.VariableStore method*), 612
add_element() (*robot.libraries.XML.XML method*), 155

add_language() (*robot.conf.languages.Languages method*), 32
add_language() (*robot.parsing.lexer.context.FileContext method*), 323
add_language() (*robot.parsing.lexer.context.InitFileContext method*), 324
add_language() (*robot.parsing.lexer.context.ResourceFileContext method*), 324
add_language() (*robot.parsing.lexer.context.TestCaseFileContext method*), 324
add_path() (*in module robot.pythonpathsetter*), 620
add_result() (*robot.result.executionresult.CombinedResult method*), 414
add_stat() (*robot.model.stats.SuiteStat method*), 275
add_tags (*robot.model.configurer.SuiteConfigurer attribute*), 234
add_tags (*robot.result.configurer.SuiteConfigurer attribute*), 408
add_test() (*robot.model.stats.CombinedTagStat method*), 276
add_test() (*robot.model.stats.Stat method*), 275
add_test() (*robot.model.stats.SuiteStat method*), 275
add_test() (*robot.model.stats.TagStat method*), 276
add_test() (*robot.model.stats.TotalStat method*), 275
add_test() (*robot.model.suitestatistics.SuiteStatisticsBuilder method*), 277
add_test() (*robot.model.tagstatistics.TagStatisticsBuilder method*), 282
add_test() (*robot.model.totalstatistics.TotalStatistics method*), 288
add_test() (*robot.model.totalstatistics.TotalStatisticsBuilder method*), 288
add_time_to_date() (*in module robot.libraries.DateTime*), 110
add_time_to_time() (*in module robot.libraries.DateTime*), 111
addFilter() (*robot.output.pyloggingconf.RobotHandler method*), 312
after (*robot.variables.search.VariableMatch attribute*), 611
after() (*robot.libraries.dialogs_py.InputDialog method*), 171
after() (*robot.libraries.dialogs_py.MessageDialog method*), 157
after() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 199
after() (*robot.libraries.dialogs_py.PassFailDialog method*), 213
after() (*robot.libraries.dialogs_py.SelectionDialog method*), 185
after_cancel() (*robot.libraries.dialogs_py.InputDialog method*), 172
after_cancel() (*robot.libraries.dialogs_py.MessageDialog method*), 158
after_cancel() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 200
after_cancel() (*robot.libraries.dialogs_py.PassFailDialog method*), 214
after_cancel() (*robot.libraries.dialogs_py.SelectionDialog method*), 186
after_idle() (*robot.libraries.dialogs_py.InputDialog method*), 172
after_idle() (*robot.libraries.dialogs_py.MessageDialog method*), 158
after_idle() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 200
after_idle() (*robot.libraries.dialogs_py.PassFailDialog method*), 214
after_idle() (*robot.libraries.dialogs_py.SelectionDialog method*), 186
alias (*robot.parsing.model.statements.LibraryImport attribute*), 351
aliases (*robot.parsing.lexer.settings.InitFileSettings attribute*), 326
aliases (*robot.parsing.lexer.settings.KeywordSettings attribute*), 327
aliases (*robot.parsing.lexer.settings.ResourceFileSettings attribute*), 326
aliases (*robot.parsing.lexer.settings.Settings attribute*), 326
aliases (*robot.parsing.lexer.settings.TestCaseFileSettings attribute*), 326
aliases (*robot.parsing.lexer.settings.TestCaseSettings attribute*), 327
aliases (*robot.running.arguments.typeconverters.BooleanConverter attribute*), 533
aliases (*robot.running.arguments.typeconverters.ByteArrayConverter attribute*), 535
aliases (*robot.running.arguments.typeconverters.BytesConverter attribute*), 535
aliases (*robot.running.arguments.typeconverters.CombinedConverter attribute*), 540
aliases (*robot.running.arguments.typeconverters.CustomConverter attribute*), 540
aliases (*robot.running.arguments.typeconverters.DateConverter attribute*), 536
aliases (*robot.running.arguments.typeconverters.DateTimeConverter attribute*), 535
aliases (*robot.running.arguments.typeconverters.DecimalConverter attribute*), 534
aliases (*robot.running.arguments.typeconverters.DictionaryConverter attribute*), 539
aliases (*robot.running.arguments.typeconverters.EnumConverter attribute*), 532
aliases (*robot.running.arguments.typeconverters.FloatConverter attribute*), 534
aliases (*robot.running.arguments.typeconverters.FrozenSetConverter attribute*), 540

aliases (*robot.running.arguments.typeconverters.IntegerConverter attribute*), 533
aliases (*robot.running.arguments.typeconverters.ListConverter attribute*), 538
aliases (*robot.running.arguments.typeconverters.NoneConverter attribute*), 537
aliases (*robot.running.arguments.typeconverters.PathConverter attribute*), 537
aliases (*robot.running.arguments.typeconverters.SetConverter attribute*), 539
aliases (*robot.running.arguments.typeconverters.StringConverter attribute*), 532
aliases (*robot.running.arguments.typeconverters.TimeDeltaConverter attribute*), 536
aliases (*robot.running.arguments.typeconverters.TupleConverter attribute*), 538
aliases (*robot.running.arguments.typeconverters.TypeConverter attribute*), 531
aliases (*robot.running.arguments.typeconverters.TypedDictConverter attribute*), 538
all (*robot.model.keyword.Keywords attribute*), 261
all_tags (*robot.libdocpkg.model.LibraryDoc attribute*), 71
AllKeywordsRemover (*class in robot.result.keywordremover*), 415
ALLOW_VARIABLES (*robot.parsing.lexer.tokens.END attribute*), 338
ALLOW_VARIABLES (*robot.parsing.lexer.tokens.EOS attribute*), 336
ALLOW_VARIABLES (*robot.parsing.lexer.tokens.Token attribute*), 335
ALLOWED_TYPES (*robot.running.model.Import attribute*), 576
also_teardown_message (*robot.running.status.ParentMessage attribute*), 584
also_teardown_message (*robot.running.status.SuiteMessage attribute*), 584
also_teardown_message (*robot.running.status.TestMessage attribute*), 584
also_teardown_skip_message (*robot.running.status.ParentMessage attribute*), 584
also_teardown_skip_message (*robot.running.status.SuiteMessage attribute*), 584
also_teardown_skip_message (*robot.running.status.TestMessage attribute*), 584
anchor() (*robot.libraries.dialogs_py.InputDialog method*), 172
anchor() (*robot.libraries.dialogs_py.MessageDialog method*), 158
anchor() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 200
anchor() (*robot.libraries.dialogs_py.PassFailDialog method*), 214
anchor() (*robot.libraries.dialogs_py.SelectionDialog method*), 186
and_prefixes (*robot.conf.languages.Bg attribute*), 60
and_prefixes (*robot.conf.languages.Bs attribute*), 39
and_prefixes (*robot.conf.languages.Cs attribute*), 36
and_prefixes (*robot.conf.languages.De attribute*), 43
and_prefixes (*robot.conf.languages.En attribute*), 35
and_prefixes (*robot.conf.languages.Es attribute*), 52
and_prefixes (*robot.conf.languages.Fi attribute*), 40
and_prefixes (*robot.conf.languages.Fr attribute*), 42
and_prefixes (*robot.conf.languages.Hi attribute*), 64
and_prefixes (*robot.conf.languages.It attribute*), 63
and_prefixes (*robot.conf.languages.Language attribute*), 33
and_prefixes (*robot.conf.languages.Nl attribute*), 38
and_prefixes (*robot.conf.languages.Pl attribute*), 49
and_prefixes (*robot.conf.languages.Pt attribute*), 46
and_prefixes (*robot.conf.languages.PtBr attribute*), 45
and_prefixes (*robot.conf.languages.Ro attribute*), 61
and_prefixes (*robot.conf.languages.Ru attribute*), 53
and_prefixes (*robot.conf.languages.Sv attribute*), 58
and_prefixes (*robot.conf.languages.Th attribute*), 47
and_prefixes (*robot.conf.languages.Tr attribute*), 57
and_prefixes (*robot.conf.languages.Uk attribute*), 50
and_prefixes (*robot.conf.languages.ZhCn attribute*), 54
and_prefixes (*robot.conf.languages.ZhTw attribute*), 56
AndTagPattern (*class in robot.model.tags*), 277
AnsiHighlighter (*class in robot.output.console.highlighting*), 303
any_timeout_occurred() (*robot.running.timeouts.TestTimeout method*), 548
append() (*robot.model.body.BaseBody method*), 230
append() (*robot.model.body.Body method*), 230
append() (*robot.model.body.Branches method*), 232
append() (*robot.model.itemlist.ItemList method*), 259
append() (*robot.model.keyword.Keywords method*), 261
append() (*robot.model.message.Messages method*), 264
append() (*robot.model.testcase.TestCases method*), 284
append() (*robot.model.testsuite.TestSuites method*), 287
append() (*robot.result.model.Body method*), 458
append() (*robot.result.model.Branches method*), 459

append() (*robot.result.model.Iterations method*), 461
append() (*robot.running.model.Body method*), 553
append() (*robot.running.model.Imports method*), 576
append_to_environment_variable() (*robot.libraries.OperatingSystem.OperatingSystem method*), 119
append_to_file() (*robot.libraries.OperatingSystem.OperatingSystem method*), 117
append_to_list() (*robot.libraries.Collections.Collections method*), 102
Application (*class in robot.utils.application*), 591
ArgFileParser (*class in robot.utils.argumentparser*), 592
ArgInfo (*class in robot.running.arguments.argumentspec*), 530
ArgLimitValidator (*class in robot.utils.argumentparser*), 592
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*), 617
args (*robot.errors.ContinueLoop attribute*), 617
args (*robot.errors.DataError attribute*), 613
args (*robot.errors.ExecutionFailed attribute*), 615
args (*robot.errors.ExecutionFailures attribute*), 615
args (*robot.errors.ExecutionPassed attribute*), 616
args (*robot.errors.ExecutionStatus attribute*), 615
args (*robot.errors.FrameworkError attribute*), 613
args (*robot.errors.HandlerExecutionFailed attribute*), 615
args (*robot.errors.Information attribute*), 614
args (*robot.errors.KeywordError attribute*), 614
args (*robot.errors.PassExecution attribute*), 617
args (*robot.errors.RemoteError attribute*), 618
args (*robot.errors.ReturnFromKeyword attribute*), 618
args (*robot.errors.RobotError attribute*), 613
args (*robot.errors.TimeoutError attribute*), 614
args (*robot.errors.UserKeywordExecutionFailed attribute*), 616
args (*robot.errors.VariableError attribute*), 614
args (*robot.libraries.BuiltIn.RobotNotRunningError attribute*), 99
args (*robot.libraries.Telnet.NoMatchError attribute*), 146
args (*robot.model.keyword.Keyword attribute*), 259
args (*robot.parsing.model.statements.Fixture attribute*), 349
args (*robot.parsing.model.statements.KeywordCall attribute*), 370
args (*robot.parsing.model.statements.LibraryImport attribute*), 351
args (*robot.parsing.model.statements.Setup attribute*), 364
args (*robot.parsing.model.statements.SuiteSetup attribute*), 357
args (*robot.parsing.model.statements.SuiteTeardown attribute*), 358
args (*robot.parsing.model.statements.Teardown attribute*), 365
args (*robot.parsing.model.statements.TemplateArguments attribute*), 371
args (*robot.parsing.model.statements.TestSetup attribute*), 359
args (*robot.parsing.model.statements.TestTeardown attribute*), 360
args (*robot.parsing.model.statements.VariablesImport attribute*), 352
args (*robot.result.model.Break attribute*), 488
args (*robot.result.model.Continue attribute*), 485
args (*robot.result.model.For attribute*), 468
args (*robot.result.model.ForIteration attribute*), 466
args (*robot.result.model.If attribute*), 477
args (*robot.result.model.IfBranch attribute*), 475
args (*robot.result.model.Keyword attribute*), 491
args (*robot.result.model.Return attribute*), 483
args (*robot.result.model.Try attribute*), 481
args (*robot.result.model.TryBranch attribute*), 479
args (*robot.result.model.While attribute*), 472
args (*robot.result.model.WhileIteration attribute*), 470
args (*robot.result.modeldeprecation.DeprecatedAttributesMixin attribute*), 500
args (*robot.running.model.Keyword attribute*), 556
ARGUMENT (*robot.parsing.lexer.tokens.END attribute*), 338
ARGUMENT (*robot.parsing.lexer.tokens.EOS attribute*), 336
ARGUMENT (*robot.parsing.lexer.tokens.Token attribute*), 334
argument_names (*robot.running.arguments.argumentspec.ArgumentSpe attribute*), 530
ArgumentCoercer (*class in robot.libraries.Remote*), 128
ArgumentConverter (*class in robot.running.arguments.argumentconverter*), 528
ArgumentHandler (*class in robot.result.xmlelementhandlers*), 525
ArgumentMapper (*class in robot.running.arguments.argumentmapper*), 528
ArgumentParser (*class in robot.utils.argumentparser*), 592
ArgumentResolver (*class in robot.running.arguments.argumentresolver*), 529

Arguments (*class in robot.parsing.model.statements*), 368
ARGUMENTS (*robot.parsing.lexer.tokens.END attribute*), 338
ARGUMENTS (*robot.parsing.lexer.tokens.EOS attribute*), 336
ARGUMENTS (*robot.parsing.lexer.tokens.Token attribute*), 334
arguments (*robot.running.userkeywordrunner.EmbeddedArguments attribute*), 590
arguments (*robot.running.userkeywordrunner.UserKeywordRunner attribute*), 590
arguments_setting (*robot.conf.languages.Bg attribute*), 60
arguments_setting (*robot.conf.languages.Bs attribute*), 39
arguments_setting (*robot.conf.languages.Cs attribute*), 36
arguments_setting (*robot.conf.languages.De attribute*), 43
arguments_setting (*robot.conf.languages.En attribute*), 35
arguments_setting (*robot.conf.languages.Es attribute*), 51
arguments_setting (*robot.conf.languages.Fi attribute*), 40
arguments_setting (*robot.conf.languages.Fr attribute*), 42
arguments_setting (*robot.conf.languages.Hi attribute*), 64
arguments_setting (*robot.conf.languages.It attribute*), 63
arguments_setting (*robot.conf.languages.Language attribute*), 33
arguments_setting (*robot.conf.languages.Nl attribute*), 37
arguments_setting (*robot.conf.languages.Pl attribute*), 49
arguments_setting (*robot.conf.languages.Pt attribute*), 46
arguments_setting (*robot.conf.languages.PtBr attribute*), 44
arguments_setting (*robot.conf.languages.Ro attribute*), 61
arguments_setting (*robot.conf.languages.Ru attribute*), 53
arguments_setting (*robot.conf.languages.Sv attribute*), 58
arguments_setting (*robot.conf.languages.Th attribute*), 47
arguments_setting (*robot.conf.languages.Tr attribute*), 57
arguments_setting (*robot.conf.languages.Uk attribute*), 50
arguments_setting (*robot.conf.languages.ZhCn attribute*), 54
arguments_setting (*robot.conf.languages.ZhTw attribute*), 56
ArgumentsHandler (*class in robot.result.xmlelementhandlers*), 525
ArgumentSpec (*class in robot.running.arguments.argumentspec*), 530
ArgumentValidator (*class in robot.running.arguments.argumentvalidator*), 531
AS (*robot.parsing.lexer.tokens.END attribute*), 338
AS (*robot.parsing.lexer.tokens.EOS attribute*), 336
AS (*robot.parsing.lexer.tokens.Token attribute*), 335
as_dict() (*robot.variables.scopes.GlobalVariables method*), 610
as_dict() (*robot.variables.scopes.VariableScopes method*), 610
as_dict() (*robot.variables.store.VariableStore method*), 612
as_dict() (*robot.variables.variables.Variables method*), 613
aspect() (*robot.libraries.dialogs_py.InputDialog method*), 172
aspect() (*robot.libraries.dialogs_py.MessageDialog method*), 158
aspect() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 200
aspect() (*robot.libraries.dialogs_py.PassFailDialog method*), 214
aspect() (*robot.libraries.dialogs_py.SelectionDialog method*), 186
assert_almost_equal() (*in module robot.utils.asserts*), 594
assert_equal() (*in module robot.utils.asserts*), 594
assert_false() (*in module robot.utils.asserts*), 594
assert_none() (*in module robot.utils.asserts*), 594
assert_not_almost_equal() (*in module robot.utils.asserts*), 595
assert_not_equal() (*in module robot.utils.asserts*), 594
assert_not_none() (*in module robot.utils.asserts*), 594
assert_raises() (*in module robot.utils.asserts*), 594
assert_raises_with_msg() (*in module robot.utils.asserts*), 594
assert_true() (*in module robot.utils.asserts*), 594
assign (*robot.model.keyword.Keyword attribute*), 259
ASSIGN (*robot.parsing.lexer.tokens.END attribute*), 338
ASSIGN (*robot.parsing.lexer.tokens.EOS attribute*), 336
ASSIGN (*robot.parsing.lexer.tokens.Token attribute*), 334

assign (*robot.parsing.model.blocks.If attribute*), 343
assign (*robot.parsing.model.statements.ElseHeader attribute*), 376
assign (*robot.parsing.model.statements.ElseIfHeader attribute*), 375
assign (*robot.parsing.model.statements.IfElseHeader attribute*), 373
assign (*robot.parsing.model.statements.IfHeader attribute*), 374
assign (*robot.parsing.model.statements.InlineIfHeader attribute*), 374
assign (*robot.parsing.model.statements.KeywordCall attribute*), 370
assign (*robot.result.model.Break attribute*), 488
assign (*robot.result.model.Continue attribute*), 486
assign (*robot.result.model.For attribute*), 468
assign (*robot.result.model.ForIteration attribute*), 466
assign (*robot.result.model.If attribute*), 477
assign (*robot.result.model.IfBranch attribute*), 475
assign (*robot.result.model.Keyword attribute*), 491
assign (*robot.result.model.Return attribute*), 484
assign (*robot.result.model.Try attribute*), 481
assign (*robot.result.model.TryBranch attribute*), 479
assign (*robot.result.model.While attribute*), 472
assign (*robot.result.model.WhileIteration attribute*), 470
assign (*robot.result.modeldeprecation.DeprecatedAttributesMixin attribute*), 500
assign (*robot.running.model.Keyword attribute*), 556
assign() (*robot.variables.assigner.VariableAssigner method*), 606
assigner() (*robot.variables.assigner.VariableAssignment method*), 606
AssignHandler (*class in robot.result.xmlelementhandlers*), 525
AssignmentValidator (*class in robot.variables.assigner*), 606
attribute_escape() (*in module robot.utils.markuputils*), 602
attributes() (*robot.libraries.dialogs_py.InputDialog method*), 172
attributes() (*robot.libraries.dialogs_py.MessageDialog method*), 158
attributes() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 200
attributes() (*robot.libraries.dialogs_py.PassFailDialog method*), 214
attributes() (*robot.libraries.dialogs_py.SelectionDialog method*), 186

B

BaseBody (*class in robot.model.body*), 229
BaseParser (*class in robot.running.builder.parsers*), 542
bbox() (*robot.libraries.dialogs_py.InputDialog method*), 172
bbox() (*robot.libraries.dialogs_py.MessageDialog method*), 158
bbox() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 200
bbox() (*robot.libraries.dialogs_py.PassFailDialog method*), 214
bbox() (*robot.libraries.dialogs_py.SelectionDialog method*), 186
bdd_prefixes (*robot.conf.languages.Bg attribute*), 60
bdd_prefixes (*robot.conf.languages.Bs attribute*), 39
bdd_prefixes (*robot.conf.languages.Cs attribute*), 36
bdd_prefixes (*robot.conf.languages.De attribute*), 43
bdd_prefixes (*robot.conf.languages.En attribute*), 35
bdd_prefixes (*robot.conf.languages.Es attribute*), 52
bdd_prefixes (*robot.conf.languages.Fi attribute*), 40
bdd_prefixes (*robot.conf.languages.Fr attribute*), 42
bdd_prefixes (*robot.conf.languages.Hi attribute*), 64
bdd_prefixes (*robot.conf.languages.It attribute*), 63
bdd_prefixes (*robot.conf.languages.Language attribute*), 34
bdd_prefixes (*robot.conf.languages.Nl attribute*), 38
bdd_prefixes (*robot.conf.languages.Pl attribute*), 49
bdd_prefixes (*robot.conf.languages.Pt attribute*), 46
bdd_prefixes (*robot.conf.languages.PtBr attribute*), 45
bdd_prefixes (*robot.conf.languages.Ro attribute*), 61
bdd_prefixes (*robot.conf.languages.Ru attribute*), 53
bdd_prefixes (*robot.conf.languages.Sv attribute*), 59
bdd_prefixes (*robot.conf.languages.Th attribute*), 47
bdd_prefixes (*robot.conf.languages.Tr attribute*), 57
bdd_prefixes (*robot.conf.languages.Uk attribute*), 50
bdd_prefixes (*robot.conf.languages.ZhCn attribute*), 54
bdd_prefixes (*robot.conf.languages.ZhTw attribute*), 56
before (*robot.variables.search.VariableMatch attribute*), 611
bell() (*robot.libraries.dialogs_py.InputDialog method*), 172
bell() (*robot.libraries.dialogs_py.MessageDialog method*), 158
bell() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 200
bell() (*robot.libraries.dialogs_py.PassFailDialog method*), 214
bell() (*robot.libraries.dialogs_py.SelectionDialog method*), 186
Bg (*class in robot.conf.languages*), 59
binary (*robot.libraries.Remote.ArgumentCoercer attribute*), 129
bind() (*robot.libraries.dialogs_py.InputDialog method*), 172

bind() (*robot.libraries.dialogs_py.MessageDialog method*), 158
bind() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 200
bind() (*robot.libraries.dialogs_py.PassFailDialog method*), 214
bind() (*robot.libraries.dialogs_py.SelectionDialog method*), 186
bind_all() (*robot.libraries.dialogs_py.InputDialog method*), 173
bind_all() (*robot.libraries.dialogs_py.MessageDialog method*), 159
bind_all() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 201
bind_all() (*robot.libraries.dialogs_py.PassFailDialog method*), 215
bind_all() (*robot.libraries.dialogs_py.SelectionDialog method*), 187
bind_class() (*robot.libraries.dialogs_py.InputDialog method*), 173
bind_class() (*robot.libraries.dialogs_py.MessageDialog method*), 159
bind_class() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 201
bind_class() (*robot.libraries.dialogs_py.PassFailDialog method*), 215
bind_class() (*robot.libraries.dialogs_py.SelectionDialog method*), 187
bindtags() (*robot.libraries.dialogs_py.InputDialog method*), 173
bindtags() (*robot.libraries.dialogs_py.MessageDialog method*), 159
bindtags() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 201
bindtags() (*robot.libraries.dialogs_py.PassFailDialog method*), 215
bindtags() (*robot.libraries.dialogs_py.SelectionDialog method*), 187
bit_length() (*robot.reporting.stringcache.StringIndex method*), 401
Block (*class in robot.parsing.model.blocks*), 340
BlockLexer (*class in robot.parsing.lexer.blocklexers*), 318
BlockParser (*class in robot.parsing.parser.blockparsers*), 389
Body (*class in robot.model.body*), 230
Body (*class in robot.result.model*), 458
Body (*class in robot.running.model*), 553
body (*robot.model.control.For attribute*), 239
body (*robot.model.control.If attribute*), 242
body (*robot.model.control.IfBranch attribute*), 241
body (*robot.model.control.Try attribute*), 245
body (*robot.model.control.TryBranch attribute*), 244
body (*robot.model.control.While attribute*), 240
body (*robot.model.testcase.TestCase attribute*), 283
body (*robot.result.model.Break attribute*), 488
body (*robot.result.model.Continue attribute*), 485
body (*robot.result.model.For attribute*), 467
body (*robot.result.model.ForIteration attribute*), 465
body (*robot.result.model.If attribute*), 477
body (*robot.result.model.IfBranch attribute*), 475
body (*robot.result.model.Keyword attribute*), 490
body (*robot.result.model.Return attribute*), 483
body (*robot.result.model.TestCase attribute*), 494
body (*robot.result.model.Try attribute*), 481
body (*robot.result.model.TryBranch attribute*), 479
body (*robot.result.model.While attribute*), 472
body (*robot.result.model.WhileIteration attribute*), 469
body (*robot.running.model.For attribute*), 558
body (*robot.running.model.If attribute*), 562
body (*robot.running.model.IfBranch attribute*), 561
body (*robot.running.model.TestCase attribute*), 570
body (*robot.running.model.Try attribute*), 565
body (*robot.running.model.TryBranch attribute*), 564
body (*robot.running.model.UserKeyword attribute*), 576
body (*robot.running.model.While attribute*), 560
body_class (*robot.model.control.For attribute*), 238
body_class (*robot.model.control.IfBranch attribute*), 241
body_class (*robot.model.control.TryBranch attribute*), 244
body_class (*robot.model.control.While attribute*), 240
body_class (*robot.model.testcase.TestCase attribute*), 282
body_class (*robot.result.model.Break attribute*), 487
body_class (*robot.result.model.Continue attribute*), 485
body_class (*robot.result.model.For attribute*), 468
body_class (*robot.result.model.ForIteration attribute*), 465
body_class (*robot.result.model.IfBranch attribute*), 474
body_class (*robot.result.model.Keyword attribute*), 490
body_class (*robot.result.model.Return attribute*), 483
body_class (*robot.result.model.TestCase attribute*), 493
body_class (*robot.result.model.TryBranch attribute*), 478
body_class (*robot.result.model.While attribute*), 472
body_class (*robot.result.model.WhileIteration attribute*), 469
body_class (*robot.running.model.For attribute*), 557
body_class (*robot.running.model.IfBranch attribute*), 560
body_class (*robot.running.model.TestCase attribute*), 570

body_class (*robot.running.model.TryBranch attribute*), 563
body_class (*robot.running.model.While attribute*), 559
BodyItem (*class in robot.model.body*), 228
BodyRunner (*class in robot.running.bodyrunner*), 549
BooleanConverter (*class in robot.running.arguments.typeconverters*), 533
branch_class (*robot.model.body.Branches attribute*), 232
branch_class (*robot.model.control.If attribute*), 242
branch_class (*robot.model.control.Try attribute*), 245
branch_class (*robot.result.model.Branches attribute*), 459
branch_class (*robot.result.model.If attribute*), 476
branch_class (*robot.result.model.Try attribute*), 480
branch_class (*robot.running.model.If attribute*), 562
branch_class (*robot.running.model.Try attribute*), 564
Branches (*class in robot.model.body*), 232
Branches (*class in robot.result.model*), 459
branches_class (*robot.model.control.If attribute*), 242
branches_class (*robot.model.control.Try attribute*), 245
branches_class (*robot.result.model.If attribute*), 476
branches_class (*robot.result.model.Try attribute*), 481
branches_class (*robot.running.model.If attribute*), 562
branches_class (*robot.running.model.Try attribute*), 565
BranchHandler (*class in robot.result.xmlelementhandlers*), 521
Break (*class in robot.model.control*), 249
Break (*class in robot.parsing.model.statements*), 384
Break (*class in robot.result.model*), 487
Break (*class in robot.running.model*), 568
BREAK (*robot.model.body.BodyItem attribute*), 228
BREAK (*robot.model.control.Break attribute*), 249
BREAK (*robot.model.control.Continue attribute*), 248
BREAK (*robot.model.control.For attribute*), 239
BREAK (*robot.model.control.If attribute*), 243
BREAK (*robot.model.control.IfBranch attribute*), 241
BREAK (*robot.model.control.Return attribute*), 247
BREAK (*robot.model.control.Try attribute*), 245
BREAK (*robot.model.control.TryBranch attribute*), 244
BREAK (*robot.model.control.While attribute*), 240
BREAK (*robot.model.keyword.Keyword attribute*), 260
BREAK (*robot.model.message.Message attribute*), 263
BREAK (*robot.output.loggerhelper.Message attribute*), 309
BREAK (*robot.parsing.lexer.tokens.END attribute*), 338
BREAK (*robot.parsing.lexer.tokens.EOS attribute*), 336
BREAK (*robot.parsing.lexer.tokens.Token attribute*), 335
BREAK (*robot.result.model.Break attribute*), 488
BREAK (*robot.result.model.Continue attribute*), 485
BREAK (*robot.result.model.For attribute*), 467
BREAK (*robot.result.model.ForIteration attribute*), 465
BREAK (*robot.result.model.If attribute*), 476
BREAK (*robot.result.model.IfBranch attribute*), 474
BREAK (*robot.result.model.Keyword attribute*), 491
BREAK (*robot.result.model.Message attribute*), 463
BREAK (*robot.result.model.Return attribute*), 483
BREAK (*robot.result.model.Try attribute*), 481
BREAK (*robot.result.model.TryBranch attribute*), 478
BREAK (*robot.result.model.While attribute*), 472
BREAK (*robot.result.model.WhileIteration attribute*), 470
BREAK (*robot.running.model.Break attribute*), 569
BREAK (*robot.running.model.Continue attribute*), 567
BREAK (*robot.running.model.For attribute*), 558
BREAK (*robot.running.model.If attribute*), 562
BREAK (*robot.running.model.IfBranch attribute*), 560
BREAK (*robot.running.model.Keyword attribute*), 555
BREAK (*robot.running.model.Return attribute*), 566
BREAK (*robot.running.model.Try attribute*), 565
BREAK (*robot.running.model.TryBranch attribute*), 563
BREAK (*robot.running.model.While attribute*), 559
break_class (*robot.model.body.BaseBody attribute*), 229
break_class (*robot.model.body.Body attribute*), 231
break_class (*robot.model.body.Branches attribute*), 232
break_class (*robot.result.model.Body attribute*), 458
break_class (*robot.result.model.Branches attribute*), 459
break_class (*robot.result.model.Iterations attribute*), 461
break_class (*robot.running.model.Body attribute*), 554
BreakHandler (*class in robot.result.xmlelementhandlers*), 522
BreakLexer (*class in robot.parsing.lexer.statementlexers*), 333
BreakLoop, 617
Bs (*class in robot.conf.languages*), 38
build() (*robot.libdocpkg.builder.DocumentationBuilder method*), 69
build() (*robot.libdocpkg.jsonbuilder.JsonDocBuilder method*), 71
build() (*robot.libdocpkg.robotbuilder.LibraryDocBuilder method*), 72
build() (*robot.libdocpkg.robotbuilder.ResourceDocBuilder method*), 72

build() (*robot.libdocpkg.robotbuilder.SuiteDocBuilder method*), 72
build() (*robot.libdocpkg.xmlbuilder.XmlDocBuilder method*), 73
build() (*robot.parsing.suitestructure.SuiteStructureBuilder method*), 393
build() (*robot.reporting.jsmodelbuilders.ErrorMessageBuilder method*), 395
build() (*robot.reporting.jsmodelbuilders.ErrorsBuilder method*), 395
build() (*robot.reporting.jsmodelbuilders.KeywordBuilder method*), 394
build() (*robot.reporting.jsmodelbuilders.MessageBuilder method*), 394
build() (*robot.reporting.jsmodelbuilders.StatisticsBuilder method*), 394
build() (*robot.reporting.jsmodelbuilders.SuiteBuilder method*), 394
build() (*robot.reporting.jsmodelbuilders.TestBuilder method*), 394
build() (*robot.result.resultbuilder.ExecutionResultBuilder method*), 501
build() (*robot.running.builder.builders.ResourceFileBuilder method*), 542
build() (*robot.running.builder.builders.TestSuiteBuilder method*), 542
build() (*robot.running.builder.transformers.ForBuilder method*), 546
build() (*robot.running.builder.transformers.IfBuilder method*), 546
build() (*robot.running.builder.transformers.TryBuilder method*), 547
build() (*robot.running.builder.transformers.WhileBuilder method*), 547
build_from() (*robot.reporting.jsmodelbuilders.JsModelBuilder method*), 394
build_from_dict() (*robot.libdocpkg.jsonbuilder.JsonDocBuilder method*), 71
build_keyword() (*robot.libdocpkg.robotbuilder.KeywordDocBuilder method*), 72
build_keyword() (*robot.reporting.jsmodelbuilders.KeywordBuilder method*), 394
build_keywords() (*robot.libdocpkg.robotbuilder.KeywordDocBuilder method*), 72
build_suite() (*robot.running.builder.parsers.RestParser method*), 543
build_suite() (*robot.running.builder.parsers.RobotParser method*), 543
BuiltIn (*class in robot.libraries.BuiltIn*), 73
but_prefixes (*robot.conf.languages.Bg attribute*), 60
but_prefixes (*robot.conf.languages.Bs attribute*), 39
but_prefixes (*robot.conf.languages.Cs attribute*), 36
but_prefixes (*robot.conf.languages.De attribute*), 43
but_prefixes (*robot.conf.languages.En attribute*), 35
but_prefixes (*robot.conf.languages.Es attribute*), 52
but_prefixes (*robot.conf.languages.Fi attribute*), 40
but_prefixes (*robot.conf.languages.Fr attribute*), 42
but_prefixes (*robot.conf.languages.Hi attribute*), 64
but_prefixes (*robot.conf.languages.It attribute*), 63
but_prefixes (*robot.conf.languages.Language attribute*), 33
but_prefixes (*robot.conf.languages.Nl attribute*), 38
but_prefixes (*robot.conf.languages.Pl attribute*), 49
but_prefixes (*robot.conf.languages.Pt attribute*), 46
but_prefixes (*robot.conf.languages.PtBr attribute*), 45
but_prefixes (*robot.conf.languages.Ro attribute*), 61
but_prefixes (*robot.conf.languages.Ru attribute*), 53
but_prefixes (*robot.conf.languages.Sv attribute*), 59
but_prefixes (*robot.conf.languages.Th attribute*), 47
but_prefixes (*robot.conf.languages.Tr attribute*), 57
but_prefixes (*robot.conf.languages.Uk attribute*), 50
but_prefixes (*robot.conf.languages.ZhCn attribute*), 54
but_prefixes (*robot.conf.languages.ZhTw attribute*), 56
by_method_name() (*robot.output.listenerarguments.EndKeywordArgu class method*), 306
by_method_name() (*robot.output.listenerarguments.EndSuiteArgument class method*), 306
by_method_name() (*robot.output.listenerarguments.EndTestArguments class method*), 306
by_method_name() (*robot.output.listenerarguments.ListenerArguments class method*), 306
by_method_name() (*robot.output.listenerarguments.MessageArgument class method*), 306
by_method_name() (*robot.output.listenerarguments.StartKeywordArgu class method*), 306
by_method_name() (*robot.output.listenerarguments.StartSuiteArgumen class method*), 306
by_method_name() (*robot.output.listenerarguments.StartTestArgument class method*), 306
ByNameKeywordRemover (*class in robot.result.keywordremover*), 424
ByPathImporter (*class in robot.utils.importer*), 601
ByTagKeywordRemover (*class in robot.result.keywordremover*), 428
ByteArrayConverter (*class in robot.running.arguments.typeconverters*), 535
BytesConverter (*class in robot.running.arguments.typeconverters*), 534

C

cache_only (*robot.output.logger.Logger attribute*), 308

call_method() (*robot.libraries.BuiltIn.BuiltIn method*), 76
called (*robot.output.listenermethods.ListenerMethod attribute*), 307
can_continue() (*robot.errors.BreakLoop method*), 617
can_continue() (*robot.errors.ContinueLoop method*), 617
can_continue() (*robot.errors.ExecutionFailed method*), 615
can_continue() (*robot.errors.ExecutionFailures method*), 616
can_continue() (*robot.errors.ExecutionPassed method*), 616
can_continue() (*robot.errors.ExecutionStatus method*), 615
can_continue() (*robot.errors.HandlerExecutionFailed method*), 615
can_continue() (*robot.errors.PassExecution method*), 617
can_continue() (*robot.errors.ReturnFromKeyword method*), 618
can_continue() (*robot.errors.UserKeywordExecutionFailed method*), 616
catenate() (*robot.libraries.BuiltIn.BuiltIn method*), 76
cget() (*robot.libraries.dialogs_py.InputDialog method*), 173
cget() (*robot.libraries.dialogs_py.MessageDialog method*), 159
cget() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 201
cget() (*robot.libraries.dialogs_py.PassFailDialog method*), 215
cget() (*robot.libraries.dialogs_py.SelectionDialog method*), 187
check_expansion() (*robot.reporting.jsbuildingcontext.JsBuildingContext method*), 394
child() (*robot.libraries.XML.Location method*), 157
children (*robot.result.model.Keyword attribute*), 490
children (*robot.result.xmlelementhandlers.ArgumentHandler attribute*), 526
children (*robot.result.xmlelementhandlers.ArgumentsHandler attribute*), 525
children (*robot.result.xmlelementhandlers.AssignHandler attribute*), 525
children (*robot.result.xmlelementhandlers.BranchHandler attribute*), 521
children (*robot.result.xmlelementhandlers.BreakHandler attribute*), 522
children (*robot.result.xmlelementhandlers.ContinueHandler attribute*), 522
children (*robot.result.xmlelementhandlers.DocHandler attribute*), 523
children (*robot.result.xmlelementhandlers.ElementHandler attribute*), 519
children (*robot.result.xmlelementhandlers.ErrorMessageHandler attribute*), 526
children (*robot.result.xmlelementhandlers.ErrorsHandler attribute*), 526
children (*robot.result.xmlelementhandlers.ForHandler attribute*), 520
children (*robot.result.xmlelementhandlers.IfHandler attribute*), 521
children (*robot.result.xmlelementhandlers.IterationHandler attribute*), 521
children (*robot.result.xmlelementhandlers.KeywordHandler attribute*), 520
children (*robot.result.xmlelementhandlers.MessageHandler attribute*), 523
children (*robot.result.xmlelementhandlers.MetadataHandler attribute*), 523
children (*robot.result.xmlelementhandlers.MetadataItemHandler attribute*), 524
children (*robot.result.xmlelementhandlers.MetaHandler attribute*), 524
children (*robot.result.xmlelementhandlers.PatternHandler attribute*), 522
children (*robot.result.xmlelementhandlers.ReturnHandler attribute*), 522
children (*robot.result.xmlelementhandlers.RobotHandler attribute*), 519
children (*robot.result.xmlelementhandlers.RootHandler attribute*), 519
children (*robot.result.xmlelementhandlers.StatisticsHandler attribute*), 526
children (*robot.result.xmlelementhandlers.StatusHandler attribute*), 523
children (*robot.result.xmlelementhandlers.SuiteHandler attribute*), 519
children (*robot.result.xmlelementhandlers.TagHandler attribute*), 524
children (*robot.result.xmlelementhandlers.TagsHandler attribute*), 524
children (*robot.result.xmlelementhandlers.TestHandler attribute*), 520
children (*robot.result.xmlelementhandlers.TimeoutHandler attribute*), 525
children (*robot.result.xmlelementhandlers.TryHandler attribute*), 521
children (*robot.result.xmlelementhandlers.ValueHandler attribute*), 526
children (*robot.result.xmlelementhandlers.VarHandler attribute*), 525
children (*robot.result.xmlelementhandlers.WhileHandler attribute*), 520
classproperty (*class in robot.utils.misc*), 604

clear() (*robot.model.body.BaseBody method*), 230
clear() (*robot.model.body.Body method*), 231
clear() (*robot.model.body.Branches method*), 232
clear() (*robot.model.itemlist.ItemList method*), 259
clear() (*robot.model.keyword.Keywords method*), 262
clear() (*robot.model.message.Messages method*), 264
clear() (*robot.model.metadata.Metadata method*), 264
clear() (*robot.model.testcase.TestCases method*), 284
clear() (*robot.model.testsuite.TestSuites method*), 288
clear() (*robot.result.model.Body method*), 458
clear() (*robot.result.model.Branches method*), 459
clear() (*robot.result.model.Iterations method*), 461
clear() (*robot.running.model.Body method*), 554
clear() (*robot.running.model.Imports method*), 576
clear() (*robot.utils.dotdict.DotDict method*), 596
clear() (*robot.utils.normalizing.NormalizedDict method*), 605
clear() (*robot.variables.evaluation.EvaluationNamespace method*), 607
clear() (*robot.variables.scopes.GlobalVariables method*), 610
clear() (*robot.variables.store.VariableStore method*), 612
clear() (*robot.variables.variables.Variables method*), 613
clear_element() (*robot.libraries.XML.XML method*), 156
client() (*robot.libraries.dialogs_py.InputDialog method*), 173
client() (*robot.libraries.dialogs_py.MessageDialog method*), 159
client() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 201
client() (*robot.libraries.dialogs_py.PassFailDialog method*), 215
client() (*robot.libraries.dialogs_py.SelectionDialog method*), 187
clipboard_append() (*robot.libraries.dialogs_py.InputDialog method*), 173
clipboard_append() (*robot.libraries.dialogs_py.MessageDialog method*), 159
clipboard_append() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 201
clipboard_append() (*robot.libraries.dialogs_py.PassFailDialog method*), 215
clipboard_append() (*robot.libraries.dialogs_py.SelectionDialog method*), 187
clipboard_clear() (*robot.libraries.dialogs_py.InputDialog method*), 173
clipboard_clear() (*robot.libraries.dialogs_py.MessageDialog method*), 159
clipboard_clear() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 201
clipboard_clear() (*robot.libraries.dialogs_py.PassFailDialog method*), 215
clipboard_clear() (*robot.libraries.dialogs_py.SelectionDialog method*), 187
clipboard_get() (*robot.libraries.dialogs_py.InputDialog method*), 173
clipboard_get() (*robot.libraries.dialogs_py.MessageDialog method*), 159
clipboard_get() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 201
clipboard_get() (*robot.libraries.dialogs_py.PassFailDialog method*), 215
clipboard_get() (*robot.libraries.dialogs_py.SelectionDialog method*), 187
close() (*robot.libraries.Remote.TimeoutHTTPSTransport method*), 129
close() (*robot.libraries.Remote.TimeoutHTTPTransport method*), 129
close() (*robot.libraries.Telnet.TelnetConnection method*), 145
close() (*robot.output.filelogger.FileLogger method*), 305
close() (*robot.output.logger.Logger method*), 308
close() (*robot.output.output.Output method*), 311
close() (*robot.output.pyloggingconf.RobotHandler method*), 312
close() (*robot.output.xmllogger.XmlLogger method*), 313
close() (*robot.reporting.outputwriter.OutputWriter method*), 396
close() (*robot.utils.application.DefaultLogger method*), 592
close() (*robot.utils.markupwriters.HtmlWriter method*), 602
close() (*robot.utils.markupwriters.NullMarkupWriter method*), 603
close() (*robot.utils.markupwriters.XmlWriter method*), 603
close_all() (*robot.utils.connectioncache.ConnectionCache method*), 596
close_all_connections() (*robot.libraries.Telnet.Telnet method*), 141
close_connection() (*robot.libraries.Telnet.TelnetConnection*

method), 142
close_global_library_listeners() (*robot.running.importer.Importer method*), 552
close_streams() (*robot.libraries.Process.ExecutionResult method*), 128
cmdline2list() (*in module robot.utils.argumentparser*), 592
code (*robot.conf.languages.Bg attribute*), 60
code (*robot.conf.languages.Bs attribute*), 39
code (*robot.conf.languages.Cs attribute*), 36
code (*robot.conf.languages.De attribute*), 43
code (*robot.conf.languages.En attribute*), 35
code (*robot.conf.languages.Es attribute*), 52
code (*robot.conf.languages.Fi attribute*), 40
code (*robot.conf.languages.Fr attribute*), 42
code (*robot.conf.languages.Hi attribute*), 64
code (*robot.conf.languages.It attribute*), 63
code (*robot.conf.languages.Language attribute*), 33
code (*robot.conf.languages.Nl attribute*), 38
code (*robot.conf.languages.Pl attribute*), 49
code (*robot.conf.languages.Pt attribute*), 46
code (*robot.conf.languages.PtBr attribute*), 45
code (*robot.conf.languages.Ro attribute*), 61
code (*robot.conf.languages.Ru attribute*), 53
code (*robot.conf.languages.Sv attribute*), 59
code (*robot.conf.languages.Th attribute*), 47
code (*robot.conf.languages.Tr attribute*), 57
code (*robot.conf.languages.Uk attribute*), 50
code (*robot.conf.languages.ZhCn attribute*), 54
code (*robot.conf.languages.ZhTw attribute*), 56
coerce() (*robot.libraries.Remote.ArgumentCoercer method*), 129
col_offset (*robot.parsing.lexer.tokens.END attribute*), 340
col_offset (*robot.parsing.lexer.tokens.EOS attribute*), 337
col_offset (*robot.parsing.lexer.tokens.Token attribute*), 335
col_offset (*robot.parsing.model.blocks.Block attribute*), 340
col_offset (*robot.parsing.model.blocks.CommentSection attribute*), 342
col_offset (*robot.parsing.model.blocks.File attribute*), 341
col_offset (*robot.parsing.model.blocks.For attribute*), 344
col_offset (*robot.parsing.model.blocks.HeaderAndBody attribute*), 340
col_offset (*robot.parsing.model.blocks.If attribute*), 343
col_offset (*robot.parsing.model.blocks.Keyword attribute*), 343
col_offset (*robot.parsing.model.blocks.KeywordSection attribute*), 342
col_offset (*robot.parsing.model.blocks.Section attribute*), 341
col_offset (*robot.parsing.model.blocks.SettingSection attribute*), 341
col_offset (*robot.parsing.model.blocks.TestCase attribute*), 343
col_offset (*robot.parsing.model.blocks.TestCaseSection attribute*), 342
col_offset (*robot.parsing.model.blocks.Try attribute*), 344
col_offset (*robot.parsing.model.blocks.VariableSection attribute*), 342
col_offset (*robot.parsing.model.blocks.While attribute*), 344
col_offset (*robot.parsing.model.statements.Arguments attribute*), 369
col_offset (*robot.parsing.model.statements.Break attribute*), 384
col_offset (*robot.parsing.model.statements.Comment attribute*), 385
col_offset (*robot.parsing.model.statements.Config attribute*), 386
col_offset (*robot.parsing.model.statements.Continue attribute*), 383
col_offset (*robot.parsing.model.statements.DefaultTags attribute*), 356
col_offset (*robot.parsing.model.statements.Documentation attribute*), 353
col_offset (*robot.parsing.model.statements.DocumentationOrMetadata attribute*), 346
col_offset (*robot.parsing.model.statements.ElseHeader attribute*), 376
col_offset (*robot.parsing.model.statements.ElseIfHeader attribute*), 375
col_offset (*robot.parsing.model.statements.EmptyLine attribute*), 388
col_offset (*robot.parsing.model.statements.End attribute*), 380
col_offset (*robot.parsing.model.statements.Error attribute*), 387
col_offset (*robot.parsing.model.statements.ExceptHeader attribute*), 379
col_offset (*robot.parsing.model.statements.FinallyHeader attribute*), 379
col_offset (*robot.parsing.model.statements.Fixture attribute*), 349
col_offset (*robot.parsing.model.statements.ForceTags attribute*), 355
col_offset (*robot.parsing.model.statements.ForHeader attribute*), 372
col_offset (*robot.parsing.model.statements.IfElseHeader attribute*), 373
col_offset (*robot.parsing.model.statements.IfHeader*

attribute), 374
col_offset (*robot.parsing.model.statements.InlineIfHeader attribute*), 374
col_offset (*robot.parsing.model.statements.KeywordCall attribute*), 370
col_offset (*robot.parsing.model.statements.KeywordName attribute*), 364
col_offset (*robot.parsing.model.statements.KeywordTags attribute*), 356
col_offset (*robot.parsing.model.statements.LibraryImport attribute*), 351
col_offset (*robot.parsing.model.statements.LoopControl attribute*), 383
col_offset (*robot.parsing.model.statements.Metadata attribute*), 354
col_offset (*robot.parsing.model.statements.MultiValue attribute*), 348
col_offset (*robot.parsing.model.statements.NoArgumentHeader attribute*), 377
col_offset (*robot.parsing.model.statements.ResourceImport attribute*), 351
col_offset (*robot.parsing.model.statements.Return attribute*), 369
col_offset (*robot.parsing.model.statements.ReturnStatement attribute*), 382
col_offset (*robot.parsing.model.statements.SectionHeader attribute*), 350
col_offset (*robot.parsing.model.statements.Setup attribute*), 365
col_offset (*robot.parsing.model.statements.SingleValue attribute*), 347
col_offset (*robot.parsing.model.statements.Statement attribute*), 346
col_offset (*robot.parsing.model.statements.SuiteSetup attribute*), 357
col_offset (*robot.parsing.model.statements.SuiteTeardown attribute*), 358
col_offset (*robot.parsing.model.statements.Tags attribute*), 366
col_offset (*robot.parsing.model.statements.Teardown attribute*), 365
col_offset (*robot.parsing.model.statements.Template attribute*), 367
col_offset (*robot.parsing.model.statements.TemplateArguments attribute*), 371
col_offset (*robot.parsing.model.statements.TestCaseName attribute*), 363
col_offset (*robot.parsing.model.statements.TestSetup attribute*), 359
col_offset (*robot.parsing.model.statements.TestTeardown attribute*), 360
col_offset (*robot.parsing.model.statements.TestTemplate attribute*), 361
col_offset (*robot.parsing.model.statements.TestTimeout attribute*), 361
col_offset (*robot.parsing.model.statements.Timeout attribute*), 368
col_offset (*robot.parsing.model.statements.TryHeader attribute*), 378
col_offset (*robot.parsing.model.statements.Variable attribute*), 362
col_offset (*robot.parsing.model.statements.VariablesImport attribute*), 352
col_offset (*robot.parsing.model.statements.WhileHeader attribute*), 381
Collections (*class in robot.libraries.Collections*), 100
colormapwindows() (*robot.libraries.dialogs_py.InputDialog method*), 173
colormapwindows() (*robot.libraries.dialogs_py.MessageDialog method*), 159
colormapwindows() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 201
colormapwindows() (*robot.libraries.dialogs_py.PassFailDialog method*), 215
colormapwindows() (*robot.libraries.dialogs_py.SelectionDialog method*), 187
columnconfigure() (*robot.libraries.dialogs_py.InputDialog method*), 174
columnconfigure() (*robot.libraries.dialogs_py.MessageDialog method*), 160
columnconfigure() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 202
columnconfigure() (*robot.libraries.dialogs_py.PassFailDialog method*), 216
columnconfigure() (*robot.libraries.dialogs_py.SelectionDialog method*), 188
combine_lists() (*robot.libraries.Collections.Collections method*), 102
combined (*robot.model.stats.TagStat attribute*), 276
combined (*robot.model.tagstatistics.TagStatistics attribute*), 282
CombinedConverter (*class in robot.running.arguments.typeconverters*), 540
CombinedResult (*class in robot.result.executionresult*), 414
CombinedTagStat (*class in robot.model.stats*), 276

command() (*robot.libraries.dialogs_py.InputDialog method*), 174
command() (*robot.libraries.dialogs_py.MessageDialog method*), 160
command() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 202
command() (*robot.libraries.dialogs_py.PassFailDialog method*), 216
command() (*robot.libraries.dialogs_py.SelectionDialog method*), 188
Comment (*class in robot.parsing.model.statements*), 385
COMMENT (*robot.parsing.lexer.tokens.END attribute*), 338
COMMENT (*robot.parsing.lexer.tokens.EOS attribute*), 336
COMMENT (*robot.parsing.lexer.tokens.Token attribute*), 335
comment() (*robot.libraries.BuiltIn.BuiltIn method*), 76
COMMENT_HEADER (*robot.parsing.lexer.tokens.END attribute*), 338
COMMENT_HEADER (*robot.parsing.lexer.tokens.EOS attribute*), 336
COMMENT_HEADER (*robot.parsing.lexer.tokens.Token attribute*), 334
comment_section() (*robot.parsing.lexer.context.FileContext method*), 323
comment_section() (*robot.parsing.lexer.context.InitFileContext method*), 324
comment_section() (*robot.parsing.lexer.context.ResourceFileContext method*), 324
comment_section() (*robot.parsing.lexer.context.TestCaseFileContext method*), 324
CommentLexer (*class in robot.parsing.lexer.statementlexers*), 329
comments_header (*robot.conf.languages.Bg attribute*), 59
comments_header (*robot.conf.languages.Bs attribute*), 38
comments_header (*robot.conf.languages.Cs attribute*), 35
comments_header (*robot.conf.languages.De attribute*), 42
comments_header (*robot.conf.languages.En attribute*), 34
comments_header (*robot.conf.languages.Es attribute*), 51
comments_header (*robot.conf.languages.Fi attribute*), 40
comments_header (*robot.conf.languages.Fr attribute*), 41
comments_header (*robot.conf.languages.Hi attribute*), 63
comments_header (*robot.conf.languages.It attribute*), 62
comments_header (*robot.conf.languages.Language attribute*), 32
comments_header (*robot.conf.languages.Nl attribute*), 37
comments_header (*robot.conf.languages.Pl attribute*), 48
comments_header (*robot.conf.languages.Pt attribute*), 45
comments_header (*robot.conf.languages.PtBr attribute*), 44
comments_header (*robot.conf.languages.Ro attribute*), 60
comments_header (*robot.conf.languages.Ru attribute*), 52
comments_header (*robot.conf.languages.Sv attribute*), 58
comments_header (*robot.conf.languages.Th attribute*), 47
comments_header (*robot.conf.languages.Tr attribute*), 56
comments_header (*robot.conf.languages.Uk attribute*), 49
comments_header (*robot.conf.languages.ZhCn attribute*), 54
comments_header (*robot.conf.languages.ZhTw attribute*), 55
CommentSection (*class in robot.parsing.model.blocks*), 342
CommentSectionHeaderLexer (*class in robot.parsing.lexer.statementlexers*), 329
CommentSectionLexer (*class in robot.parsing.lexer.blocklexers*), 320
CommentSectionParser (*class in robot.parsing.parser.fileparser*), 391
compare() (*robot.libraries.XML.ElementComparator method*), 157
compress_text() (*in module robot.utils.compress*), 595
condition (*robot.model.control.IfBranch attribute*), 241
condition (*robot.model.control.While attribute*), 240
condition (*robot.parsing.model.blocks.If attribute*), 343
condition (*robot.parsing.model.blocks.While attribute*), 344
condition (*robot.parsing.model.statements.ElseHeader attribute*), 376
condition (*robot.parsing.model.statements.ElseIfHeader attribute*), 375
condition (*robot.parsing.model.statements.IfElseHeader*

attribute), 373
condition (*robot.parsing.model.statements.IfHeader attribute*), 374
condition (*robot.parsing.model.statements.InlineIfHeader attribute*), 374
condition (*robot.parsing.model.statements.WhileHeader attribute*), 381
condition (*robot.result.model.IfBranch attribute*), 475
condition (*robot.result.model.While attribute*), 472
condition (*robot.running.model.IfBranch attribute*), 561
condition (*robot.running.model.While attribute*), 560
Config (*class in robot.parsing.model.statements*), 386
CONFIG (*robot.parsing.lexer.tokens.END attribute*), 338
CONFIG (*robot.parsing.lexer.tokens.EOS attribute*), 336
CONFIG (*robot.parsing.lexer.tokens.Token attribute*), 335
config() (*robot.libraries.dialogs_py.InputDialog method*), 174
config() (*robot.libraries.dialogs_py.MessageDialog method*), 160
config() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 202
config() (*robot.libraries.dialogs_py.PassFailDialog method*), 216
config() (*robot.libraries.dialogs_py.SelectionDialog method*), 188
config() (*robot.model.body.BodyItem method*), 228
config() (*robot.model.control.Break method*), 249
config() (*robot.model.control.Continue method*), 248
config() (*robot.model.control.For method*), 239
config() (*robot.model.control.If method*), 243
config() (*robot.model.control.IfBranch method*), 242
config() (*robot.model.control.Return method*), 247
config() (*robot.model.control.Try method*), 246
config() (*robot.model.control.TryBranch method*), 244
config() (*robot.model.control.While method*), 240
config() (*robot.model.keyword.Keyword method*), 261
config() (*robot.model.message.Message method*), 263
config() (*robot.model.modelobject.ModelObject method*), 265
config() (*robot.model.testcase.TestCase method*), 284
config() (*robot.model.testsuite.TestSuite method*), 287
config() (*robot.output.loggerhelper.Message method*), 310
config() (*robot.result.model.Break method*), 488
config() (*robot.result.model.Continue method*), 486
config() (*robot.result.model.For method*), 468
config() (*robot.result.model.ForIteration method*), 466
config() (*robot.result.model.If method*), 477
config() (*robot.result.model.IfBranch method*), 475
config() (*robot.result.model.Keyword method*), 491
config() (*robot.result.model.Message method*), 463
config() (*robot.result.model.Return method*), 484
config() (*robot.result.model.TestCase method*), 494
config() (*robot.result.model.TestSuite method*), 496
config() (*robot.result.model.Try method*), 481
config() (*robot.result.model.TryBranch method*), 479
config() (*robot.result.model.While method*), 472
config() (*robot.result.model.WhileIteration method*), 470
config() (*robot.running.model.Break method*), 569
config() (*robot.running.model.Continue method*), 568
config() (*robot.running.model.For method*), 558
config() (*robot.running.model.If method*), 562
config() (*robot.running.model.IfBranch method*), 561
config() (*robot.running.model.Keyword method*), 556
config() (*robot.running.model.Return method*), 566
config() (*robot.running.model.TestCase method*), 570
config() (*robot.running.model.TestSuite method*), 574
config() (*robot.running.model.Try method*), 565
config() (*robot.running.model.TryBranch method*), 564
config() (*robot.running.model.While method*), 560
configure() (*robot.libraries.dialogs_py.InputDialog method*), 174
configure() (*robot.libraries.dialogs_py.MessageDialog method*), 160
configure() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 202
configure() (*robot.libraries.dialogs_py.PassFailDialog method*), 216
configure() (*robot.libraries.dialogs_py.SelectionDialog method*), 188
configure() (*robot.model.testsuite.TestSuite method*), 287
configure() (*robot.result.executionresult.CombinedResult method*), 414
configure() (*robot.result.executionresult.Result method*), 413
configure() (*robot.result.model.TestSuite method*), 499
configure() (*robot.running.model.TestSuite method*), 572
conjugate() (*robot.reporting.stringcache.StringIndex method*), 401
ConnectionCache (*class in robot.utils.connectioncache*), 595
console() (*in module robot.api.logger*), 15
console() (*in module robot.output.librarylogger*), 306
console() (*robot.libdoc.LibDoc method*), 619
console() (*robot.rebot.Rebot method*), 620
console() (*robot.run.RobotFramework method*), 622
console() (*robot.testdoc.TestDoc method*), 624
console() (*robot.utils.application.Application method*), 591
console_colors (*robot.conf.settings.RebotSettings*

attribute), 66
console_colors (*robot.conf.settings.RobotSettings attribute*), 65
console_decode() (*in module robot.utils.encoding*), 597
console_encode() (*in module robot.utils.encoding*), 597
console_markers (*robot.conf.settings.RobotSettings attribute*), 65
console_output_config (*robot.conf.settings.RebotSettings attribute*), 66
console_output_config (*robot.conf.settings.RobotSettings attribute*), 65
console_type (*robot.conf.settings.RobotSettings attribute*), 65
console_width (*robot.conf.settings.RobotSettings attribute*), 65
ConsoleOutput() (*in module robot.output.console*), 298
ConsoleViewer (*class in robot.libdocpkg.consoleviewer*), 69
contains_variable() (*in module robot.variables.search*), 611
content() (*robot.utils.markupwriters.HtmlWriter method*), 602
content() (*robot.utils.markupwriters.NullMarkupWriter method*), 603
content() (*robot.utils.markupwriters.XmlWriter method*), 603
ContinuableFailure, 12
CONTINUATION (*robot.parsing.lexer.tokens.END attribute*), 338
CONTINUATION (*robot.parsing.lexer.tokens.EOS attribute*), 336
CONTINUATION (*robot.parsing.lexer.tokens.Token attribute*), 335
Continue (*class in robot.model.control*), 248
Continue (*class in robot.parsing.model.statements*), 383
Continue (*class in robot.result.model*), 485
Continue (*class in robot.running.model*), 567
CONTINUE (*robot.model.body.BodyItem attribute*), 228
CONTINUE (*robot.model.control.Break attribute*), 249
CONTINUE (*robot.model.control.Continue attribute*), 248
CONTINUE (*robot.model.control.For attribute*), 239
CONTINUE (*robot.model.control.If attribute*), 243
CONTINUE (*robot.model.control.IfBranch attribute*), 241
CONTINUE (*robot.model.control.Return attribute*), 247
CONTINUE (*robot.model.control.Try attribute*), 245
CONTINUE (*robot.model.control.TryBranch attribute*), 244
CONTINUE (*robot.model.control.While attribute*), 240
CONTINUE (*robot.model.keyword.Keyword attribute*), 260
CONTINUE (*robot.model.message.Message attribute*), 263
CONTINUE (*robot.output.loggerhelper.Message attribute*), 309
CONTINUE (*robot.parsing.lexer.tokens.END attribute*), 338
CONTINUE (*robot.parsing.lexer.tokens.EOS attribute*), 336
CONTINUE (*robot.parsing.lexer.tokens.Token attribute*), 335
CONTINUE (*robot.result.model.Break attribute*), 488
CONTINUE (*robot.result.model.Continue attribute*), 486
CONTINUE (*robot.result.model.For attribute*), 467
CONTINUE (*robot.result.model.ForIteration attribute*), 465
CONTINUE (*robot.result.model.If attribute*), 476
CONTINUE (*robot.result.model.IfBranch attribute*), 474
CONTINUE (*robot.result.model.Keyword attribute*), 491
CONTINUE (*robot.result.model.Message attribute*), 463
CONTINUE (*robot.result.model.Return attribute*), 483
CONTINUE (*robot.result.model.Try attribute*), 481
CONTINUE (*robot.result.model.TryBranch attribute*), 478
CONTINUE (*robot.result.model.While attribute*), 472
CONTINUE (*robot.result.model.WhileIteration attribute*), 470
CONTINUE (*robot.running.model.Break attribute*), 569
CONTINUE (*robot.running.model.Continue attribute*), 567
CONTINUE (*robot.running.model.For attribute*), 558
CONTINUE (*robot.running.model.If attribute*), 562
CONTINUE (*robot.running.model.IfBranch attribute*), 560
CONTINUE (*robot.running.model.Keyword attribute*), 555
CONTINUE (*robot.running.model.Return attribute*), 566
CONTINUE (*robot.running.model.Try attribute*), 565
CONTINUE (*robot.running.model.TryBranch attribute*), 563
CONTINUE (*robot.running.model.While attribute*), 559
continue_class (*robot.model.body.BaseBody attribute*), 229
continue_class (*robot.model.body.Body attribute*), 231
continue_class (*robot.model.body.Branches attribute*), 232
continue_class (*robot.result.model.Body attribute*), 458
continue_class (*robot.result.model.Branches attribute*), 459
continue_class (*robot.result.model.Iterations attribute*), 461
continue_class (*robot.running.model.Body at-*

tribute), 554
continue_for_loop() (*robot.libraries.BuiltIn.BuiltIn method*), 76
continue_for_loop_if() (*robot.libraries.BuiltIn.BuiltIn method*), 77
continue_on_failure (*robot.errors.BreakLoop attribute*), 617
continue_on_failure (*robot.errors.ContinueLoop attribute*), 617
continue_on_failure (*robot.errors.ExecutionFailed attribute*), 615
continue_on_failure (*robot.errors.ExecutionFailures attribute*), 616
continue_on_failure (*robot.errors.ExecutionPassed attribute*), 616
continue_on_failure (*robot.errors.ExecutionStatus attribute*), 614
continue_on_failure (*robot.errors.HandlerExecutionFailed attribute*), 615
continue_on_failure (*robot.errors.PassExecution attribute*), 617
continue_on_failure (*robot.errors.ReturnFromKeyword attribute*), 618
continue_on_failure (*robot.errors.UserKeywordExecutionFailed attribute*), 616
ContinueHandler (*class in robot.result.xmlelementhandlers*), 522
ContinueLexer (*class in robot.parsing.lexer.statementlexers*), 333
ContinueLoop, 617
convert() (*robot.running.arguments.argumentconverter.ArgumentConverter method*), 528
convert() (*robot.running.arguments.argumentspec.ArgumentSpec method*), 530
convert() (*robot.running.arguments.typeconverters.BooleanConverter method*), 533
convert() (*robot.running.arguments.typeconverters.ByteArrayConverter method*), 535
convert() (*robot.running.arguments.typeconverters.BytesConverter method*), 535
convert() (*robot.running.arguments.typeconverters.CombinedConverter method*), 540
convert() (*robot.running.arguments.typeconverters.CustomConverter method*), 540
convert() (*robot.running.arguments.typeconverters.DateConverter method*), 536
convert() (*robot.running.arguments.typeconverters.DateTimeConverter method*), 535
convert() (*robot.running.arguments.typeconverters.DecimalConverter method*), 534
convert() (*robot.running.arguments.typeconverters.DictionaryConverter method*), 539
convert() (*robot.running.arguments.typeconverters.EnumConverter method*), 532
convert() (*robot.running.arguments.typeconverters.FloatConverter method*), 534
convert() (*robot.running.arguments.typeconverters.FrozenSetConverter method*), 540
convert() (*robot.running.arguments.typeconverters.IntegerConverter method*), 533
convert() (*robot.running.arguments.typeconverters.ListConverter method*), 538
convert() (*robot.running.arguments.typeconverters.NoneConverter method*), 537
convert() (*robot.running.arguments.typeconverters.PathConverter method*), 537
convert() (*robot.running.arguments.typeconverters.SetConverter method*), 539
convert() (*robot.running.arguments.typeconverters.StringConverter method*), 532
convert() (*robot.running.arguments.typeconverters.TimeDeltaConverter method*), 536
convert() (*robot.running.arguments.typeconverters.TupleConverter method*), 538
convert() (*robot.running.arguments.typeconverters.TypeConverter method*), 532
convert() (*robot.running.arguments.typeconverters.TypedDictConverter method*), 538
convert() (*robot.testdoc.JsonConverter method*), 624
convert_date() (*in module robot.libraries.DateTime*), 109
convert_docs_to_html() (*robot.libdocpkg.model.LibraryDoc method*), 71
convert_time() (*in module robot.libraries.DateTime*), 110
convert_to_binary() (*robot.libraries.BuiltIn.BuiltIn method*), 77
convert_to_boolean() (*robot.libraries.BuiltIn.BuiltIn method*), 77
convert_to_bytes() (*robot.libraries.BuiltIn.BuiltIn method*), 77
convert_to_dictionary() (*robot.libraries.Collections.Collections method*), 102
convert_to_hex() (*robot.libraries.BuiltIn.BuiltIn*

method), 77
convert_to_integer() (*robot.libraries.BuiltIn.BuiltIn method*), 78
convert_to_list() (*robot.libraries.Collections.Collections method*), 102
convert_to_lower_case() (*robot.libraries.String.String method*), 132
convert_to_number() (*robot.libraries.BuiltIn.BuiltIn method*), 78
convert_to_octal() (*robot.libraries.BuiltIn.BuiltIn method*), 78
convert_to_string() (*robot.libraries.BuiltIn.BuiltIn method*), 78
convert_to_title_case() (*robot.libraries.String.String method*), 132
convert_to_upper_case() (*robot.libraries.String.String method*), 132
convert_type_list_to_dict() (*robot.running.arguments.typevalidator.TypeValidator method*), 541
converter_for() (*robot.running.arguments.typeconverters.BooleanConverter class method*), 533
converter_for() (*robot.running.arguments.typeconverters.ByteArrayConverter class method*), 535
converter_for() (*robot.running.arguments.typeconverters.BytesConverter class method*), 535
converter_for() (*robot.running.arguments.typeconverters.CombinedConverter class method*), 540
converter_for() (*robot.running.arguments.typeconverters.CustomConverter class method*), 541
converter_for() (*robot.running.arguments.typeconverters.DateConverter class method*), 536
converter_for() (*robot.running.arguments.typeconverters.DatetimeConverter class method*), 536
converter_for() (*robot.running.arguments.typeconverters.DecimalConverter class method*), 534
converter_for() (*robot.running.arguments.typeconverters.DictionaryConverter class method*), 539
converter_for() (*robot.running.arguments.typeconverters.EnumConverter class method*), 532
converter_for() (*robot.running.arguments.typeconverters.FloatConverter class method*), 534
converter_for() (*robot.running.arguments.typeconverters.FrozenSetConverter class method*), 540
converter_for() (*robot.running.arguments.typeconverters.IntegerConverter class method*), 533
converter_for() (*robot.running.arguments.typeconverters.ListConverter class method*), 538
converter_for() (*robot.running.arguments.typeconverters.NoneConverter class method*), 537
converter_for() (*robot.running.arguments.typeconverters.PathConverter class method*), 537
converter_for() (*robot.running.arguments.typeconverters.SetConverter class method*), 539
converter_for() (*robot.running.arguments.typeconverters.StringConverter class method*), 532
converter_for() (*robot.running.arguments.typeconverters.TimeDeltaConverter class method*), 536
converter_for() (*robot.running.arguments.typeconverters.TupleConverter class method*), 538
converter_for() (*robot.running.arguments.typeconverters.TypeConverter class method*), 532
converter_for() (*robot.running.arguments.typeconverters.TypedDictConverter class method*), 538
ConverterInfo (*class in robot.running.arguments.customconverters*), 531
copy() (*robot.model.body.BodyItem method*), 228
copy() (*robot.model.control.Break method*), 249
copy() (*robot.model.control.Continue method*), 248
copy() (*robot.model.control.For method*), 239
copy() (*robot.model.control.If method*), 243
copy() (*robot.model.control.IfBranch method*), 242
copy() (*robot.model.control.Return method*), 247
copy() (*robot.model.control.Try method*), 246
copy() (*robot.model.control.TryBranch method*), 245
copy() (*robot.model.control.While method*), 241
copy() (*robot.model.keyword.Keyword method*), 261
copy() (*robot.model.message.Message method*), 263
copy() (*robot.model.metadata.Metadata method*), 264
copy() (*robot.model.modelobject.ModelObject method*), 265
copy() (*robot.model.testcase.TestCase method*), 284
copy() (*robot.model.testsuite.TestSuite method*), 287
copy() (*robot.output.loggerhelper.Message method*), 310
copy() (*robot.result.model.Break method*), 489
copy() (*robot.result.model.Continue method*), 486
copy() (*robot.result.model.For method*), 468
copy() (*robot.result.model.ForIteration method*), 466
copy() (*robot.result.model.If method*), 477
copy() (*robot.result.model.IfBranch method*), 475
copy() (*robot.result.model.Keyword method*), 491
copy() (*robot.result.model.Message method*), 463
copy() (*robot.result.model.Return method*), 484
copy() (*robot.result.model.TestCase method*), 494
copy() (*robot.result.model.TestSuite method*), 496
copy() (*robot.result.model.Try method*), 482
copy() (*robot.result.model.TryBranch method*), 479
copy() (*robot.result.model.While method*), 473
copy() (*robot.result.model.WhileIteration method*), 470
copy() (*robot.running.model.Break method*), 569
copy() (*robot.running.model.Continue method*), 568

copy() (*robot.running.model.For method*), 558
copy() (*robot.running.model.If method*), 562
copy() (*robot.running.model.IfBranch method*), 561
copy() (*robot.running.model.Keyword method*), 556
copy() (*robot.running.model.Return method*), 567
copy() (*robot.running.model.TestCase method*), 570
copy() (*robot.running.model.TestSuite method*), 574
copy() (*robot.running.model.Try method*), 565
copy() (*robot.running.model.TryBranch method*), 564
copy() (*robot.running.model.While method*), 560
copy() (*robot.utils.dotdict.DotDict method*), 596
copy() (*robot.utils.normalizing.NormalizedDict method*), 605
copy() (*robot.variables.scopes.GlobalVariables method*), 610
copy() (*robot.variables.variables.Variables method*), 613
copy_dictionary() (*robot.libraries.Collections.Collections method*), 102
copy_directory() (*robot.libraries.OperatingSystem.OperatingSystem method*), 118
copy_element() (*robot.libraries.XML.XML method*), 156
copy_file() (*robot.libraries.OperatingSystem.OperatingSystem method*), 118
copy_files() (*robot.libraries.OperatingSystem.OperatingSystem method*), 118
copy_list() (*robot.libraries.Collections.Collections method*), 102
count() (*robot.model.body.BaseBody method*), 230
count() (*robot.model.body.Body method*), 231
count() (*robot.model.body.Branches method*), 232
count() (*robot.model.itemlist.ItemList method*), 259
count() (*robot.model.keyword.Keywords method*), 262
count() (*robot.model.message.Messages method*), 264
count() (*robot.model.testcase.TestCases method*), 284
count() (*robot.model.testsuite.TestSuites method*), 288
count() (*robot.result.model.Body method*), 458
count() (*robot.result.model.Branches method*), 459
count() (*robot.result.model.Iterations method*), 461
count() (*robot.running.model.Body method*), 554
count() (*robot.running.model.Imports method*), 576
count_directories_in_directory() (*robot.libraries.OperatingSystem.OperatingSystem method*), 122
count_files_in_directory() (*robot.libraries.OperatingSystem.OperatingSystem method*), 122
count_items_in_directory() (*robot.libraries.OperatingSystem.OperatingSystem method*), 122
count_values_in_list() (*robot.libraries.Collections.Collections method*), 102
create (*robot.model.body.BaseBody attribute*), 229
create (*robot.model.body.Body attribute*), 231
create (*robot.model.body.Branches attribute*), 232
create (*robot.result.model.Body attribute*), 458
create (*robot.result.model.Branches attribute*), 460
create (*robot.result.model.Iterations attribute*), 461
create (*robot.running.model.Body attribute*), 554
create() (*robot.model.itemlist.ItemList method*), 259
create() (*robot.model.keyword.Keywords method*), 261
create() (*robot.model.message.Messages method*), 264
create() (*robot.model.testcase.TestCases method*), 284
create() (*robot.model.testsuite.TestSuites method*), 288
create() (*robot.running.bodyrunner.DurationLimit class method*), 550
create() (*robot.running.bodyrunner.IterationCountLimit class method*), 550
create() (*robot.running.bodyrunner.NoLimit class method*), 550
create() (*robot.running.bodyrunner.WhileLimit class method*), 550
create() (*robot.running.model.Imports method*), 576
create_binary_file() (*robot.libraries.OperatingSystem.OperatingSystem method*), 117
create_branch() (*robot.model.body.Branches method*), 232
create_branch() (*robot.result.model.Branches method*), 460
create_break() (*robot.model.body.BaseBody method*), 229
create_break() (*robot.model.body.Body method*), 231
create_break() (*robot.model.body.Branches method*), 232
create_break() (*robot.result.model.Body method*), 458
create_break() (*robot.result.model.Branches method*), 460
create_break() (*robot.result.model.Iterations method*), 461
create_break() (*robot.running.model.Body method*), 554
create_continue() (*robot.model.body.BaseBody method*), 229
create_continue() (*robot.model.body.Body method*), 231
create_continue() (*robot.model.body.Branches method*), 232
create_continue() (*robot.result.model.Body*

method), 458
create_continue() (*robot.result.model.Branches method*), 460
create_continue() (*robot.result.model.Iterations method*), 461
create_continue() (*robot.running.model.Body method*), 554
create_dictionary() (*robot.libraries.BuiltIn.BuiltIn method*), 79
create_directory() (*robot.libraries.OperatingSystem.OperatingSystem method*), 117
create_file() (*robot.libraries.OperatingSystem.OperatingSystem method*), 116
create_fixture() (*in module robot.model.fixture*), 258
create_for() (*robot.model.body.BaseBody method*), 229
create_for() (*robot.model.body.Body method*), 231
create_for() (*robot.model.body.Branches method*), 233
create_for() (*robot.result.model.Body method*), 458
create_for() (*robot.result.model.Branches method*), 460
create_for() (*robot.result.model.Iterations method*), 461
create_for() (*robot.running.model.Body method*), 554
create_if() (*robot.model.body.BaseBody method*), 229
create_if() (*robot.model.body.Body method*), 231
create_if() (*robot.model.body.Branches method*), 233
create_if() (*robot.result.model.Body method*), 458
create_if() (*robot.result.model.Branches method*), 460
create_if() (*robot.result.model.Iterations method*), 461
create_if() (*robot.running.model.Body method*), 554
create_iteration() (*robot.result.model.Iterations method*), 461
create_keyword() (*robot.model.body.BaseBody method*), 229
create_keyword() (*robot.model.body.Body method*), 231
create_keyword() (*robot.model.body.Branches method*), 233
create_keyword() (*robot.result.model.Body method*), 458
create_keyword() (*robot.result.model.Branches method*), 460
create_keyword() (*robot.result.model.Iterations method*), 461
create_keyword() (*robot.running.model.Body method*), 554
create_link_target() (*robot.reporting.jsbuildingcontext.JsBuildingContext method*), 393
create_list() (*robot.libraries.BuiltIn.BuiltIn method*), 79
create_message() (*robot.model.body.BaseBody method*), 229
create_message() (*robot.model.body.Body method*), 231
create_message() (*robot.model.body.Branches method*), 233
create_message() (*robot.result.model.Body method*), 458
create_message() (*robot.result.model.Branches method*), 460
create_message() (*robot.result.model.Iterations method*), 461
create_message() (*robot.running.model.Body method*), 554
create_return() (*robot.model.body.BaseBody method*), 229
create_return() (*robot.model.body.Body method*), 231
create_return() (*robot.model.body.Branches method*), 233
create_return() (*robot.result.model.Body method*), 458
create_return() (*robot.result.model.Branches method*), 460
create_return() (*robot.result.model.Iterations method*), 461
create_return() (*robot.running.model.Body method*), 554
create_runner() (*robot.running.handlers.EmbeddedArgumentsHandl method*), 551
create_runner() (*robot.running.usererrorhandler.UserErrorHandler method*), 589
create_runner() (*robot.running.userkeyword.EmbeddedArgumentsHa method*), 590
create_runner() (*robot.running.userkeyword.UserKeywordHandler method*), 590
create_setup() (*robot.model.keyword.Keywords method*), 261
create_teardown() (*robot.model.keyword.Keywords method*), 261
create_try() (*robot.model.body.BaseBody method*), 229
create_try() (*robot.model.body.Body method*), 231
create_try() (*robot.model.body.Branches method*), 233

create_try() (*robot.result.model.Body method*), 458
create_try() (*robot.result.model.Branches method*), 460
create_try() (*robot.result.model.Iterations method*), 461
create_try() (*robot.running.model.Body method*), 554
create_while() (*robot.model.body.BaseBody method*), 229
create_while() (*robot.model.body.Body method*), 231
create_while() (*robot.model.body.Branches method*), 233
create_while() (*robot.result.model.Body method*), 458
create_while() (*robot.result.model.Branches method*), 460
create_while() (*robot.result.model.Iterations method*), 461
create_while() (*robot.running.model.Body method*), 554
createLock() (*robot.output.pyloggingconf.RobotHandler method*), 312
critical (*robot.result.model.TestCase attribute*), 493
Cs (*class in robot.conf.languages*), 35
CssFileWriter (*class in robot.htmldata.htmlfilewriter*), 67
current (*robot.model.suitestatistics.SuiteStatisticsBuilder attribute*), 276
current (*robot.running.context.ExecutionContexts attribute*), 550
current (*robot.utils.connectioncache.ConnectionCache attribute*), 595
current (*robot.variables.scopes.VariableScopes attribute*), 609
current_index (*robot.utils.connectioncache.ConnectionCache attribute*), 595
current_output (*robot.libraries.Telnet.TerminalEmulator attribute*), 146
CUSTOM (*robot.libdocpkg.datatypes.TypeDoc attribute*), 70
CustomArgumentConverters (*class in robot.running.arguments.customconverters*), 531
CustomConverter (*class in robot.running.arguments.typeconverters*), 540

D

data (*robot.running.modelcombiner.ModelCombiner attribute*), 577
data_tokens (*robot.parsing.model.statements.Arguments attribute*), 369
data_tokens (*robot.parsing.model.statements.Break attribute*), 384
data_tokens (*robot.parsing.model.statements.Comment attribute*), 385
data_tokens (*robot.parsing.model.statements.Config attribute*), 386
data_tokens (*robot.parsing.model.statements.Continue attribute*), 383
data_tokens (*robot.parsing.model.statements.DefaultTags attribute*), 356
data_tokens (*robot.parsing.model.statements.Documentation attribute*), 353
data_tokens (*robot.parsing.model.statements.DocumentationOrMetada attribute*), 346
data_tokens (*robot.parsing.model.statements.ElseHeader attribute*), 376
data_tokens (*robot.parsing.model.statements.ElseIfHeader attribute*), 375
data_tokens (*robot.parsing.model.statements.EmptyLine attribute*), 388
data_tokens (*robot.parsing.model.statements.End attribute*), 380
data_tokens (*robot.parsing.model.statements.Error attribute*), 387
data_tokens (*robot.parsing.model.statements.ExceptHeader attribute*), 379
data_tokens (*robot.parsing.model.statements.FinallyHeader attribute*), 379
data_tokens (*robot.parsing.model.statements.Fixture attribute*), 349
data_tokens (*robot.parsing.model.statements.ForceTags attribute*), 355
data_tokens (*robot.parsing.model.statements.ForHeader attribute*), 372
data_tokens (*robot.parsing.model.statements.IfElseHeader attribute*), 373
data_tokens (*robot.parsing.model.statements.IfHeader attribute*), 374
data_tokens (*robot.parsing.model.statements.InlineIfHeader attribute*), 374
data_tokens (*robot.parsing.model.statements.KeywordCall attribute*), 370
data_tokens (*robot.parsing.model.statements.KeywordName attribute*), 364
data_tokens (*robot.parsing.model.statements.KeywordTags attribute*), 356
data_tokens (*robot.parsing.model.statements.LibraryImport attribute*), 351
data_tokens (*robot.parsing.model.statements.LoopControl attribute*), 383
data_tokens (*robot.parsing.model.statements.Metadata attribute*), 354
data_tokens (*robot.parsing.model.statements.MultiValue attribute*), 348

data_tokens (*robot.parsing.model.statements.NoArgumentHeader attribute*), 377
data_tokens (*robot.parsing.model.statements.ResourceImport attribute*), 351
data_tokens (*robot.parsing.model.statements.Return attribute*), 369
data_tokens (*robot.parsing.model.statements.ReturnStatement attribute*), 382
data_tokens (*robot.parsing.model.statements.SectionHeader attribute*), 350
data_tokens (*robot.parsing.model.statements.Setup attribute*), 365
data_tokens (*robot.parsing.model.statements.SingleValue attribute*), 347
data_tokens (*robot.parsing.model.statements.Statement attribute*), 346
data_tokens (*robot.parsing.model.statements.SuiteSetup attribute*), 357
data_tokens (*robot.parsing.model.statements.SuiteTeardown attribute*), 358
data_tokens (*robot.parsing.model.statements.Tags attribute*), 366
data_tokens (*robot.parsing.model.statements.Teardown attribute*), 365
data_tokens (*robot.parsing.model.statements.Template attribute*), 367
data_tokens (*robot.parsing.model.statements.TemplateArguments attribute*), 371
data_tokens (*robot.parsing.model.statements.TestCaseName attribute*), 363
data_tokens (*robot.parsing.model.statements.TestSetup attribute*), 359
data_tokens (*robot.parsing.model.statements.TestTeardown attribute*), 360
data_tokens (*robot.parsing.model.statements.TestTemplate attribute*), 361
data_tokens (*robot.parsing.model.statements.TestTimeout attribute*), 361
data_tokens (*robot.parsing.model.statements.Timeout attribute*), 368
data_tokens (*robot.parsing.model.statements.TryHeader attribute*), 378
data_tokens (*robot.parsing.model.statements.Variable attribute*), 362
data_tokens (*robot.parsing.model.statements.VariablesImport attribute*), 352
data_tokens (*robot.parsing.model.statements.WhileHeader attribute*), 381
DataError, 613
DateConverter (*class in robot.running.arguments.typeconverters*), 536
DateTimeConverter (*class in robot.running.arguments.typeconverters*), 535
De (*class in robot.conf.languages*), 42
debug() (*in module robot.api.logger*), 15
debug() (*in module robot.output.librarylogger*), 305
debug() (*robot.output.filelogger.FileLogger method*), 305
debug() (*robot.output.logger.Logger method*), 309
debug() (*robot.output.loggerhelper.AbstractLogger method*), 309
debug() (*robot.output.output.Output method*), 311
debug() (*robot.utils.importer.NoLogger method*), 602
debug_file (*robot.conf.settings.RobotSettings attribute*), 65
DebugFile() (*in module robot.output.debugfile*), 305
DecimalConverter (*class in robot.running.arguments.typeconverters*), 534
decode_bytes_to_string() (*robot.libraries.String.String method*), 133
deepcopy() (*robot.model.body.BodyItem method*), 229
deepcopy() (*robot.model.control.Break method*), 250
deepcopy() (*robot.model.control.Continue method*), 248
deepcopy() (*robot.model.control.For method*), 239
deepcopy() (*robot.model.control.If method*), 243
deepcopy() (*robot.model.control.IfBranch method*), 242
deepcopy() (*robot.model.control.Return method*), 247
deepcopy() (*robot.model.control.Try method*), 246
deepcopy() (*robot.model.control.TryBranch method*), 245
deepcopy() (*robot.model.control.While method*), 241
deepcopy() (*robot.model.keyword.Keyword method*), 261
deepcopy() (*robot.model.message.Message method*), 263
deepcopy() (*robot.model.modelobject.ModelObject method*), 265
deepcopy() (*robot.model.testcase.TestCase method*), 284
deepcopy() (*robot.model.testsuite.TestSuite method*), 287
deepcopy() (*robot.output.loggerhelper.Message method*), 310
deepcopy() (*robot.result.model.Break method*), 489
deepcopy() (*robot.result.model.Continue method*), 486
deepcopy() (*robot.result.model.For method*), 468
deepcopy() (*robot.result.model.ForIteration method*), 466
deepcopy() (*robot.result.model.If method*), 477
deepcopy() (*robot.result.model.IfBranch method*), 475

deepcopy() (*robot.result.model.Keyword method*), 492
deepcopy() (*robot.result.model.Message method*), 463
deepcopy() (*robot.result.model.Return method*), 484
deepcopy() (*robot.result.model.TestCase method*), 494
deepcopy() (*robot.result.model.TestSuite method*), 496
deepcopy() (*robot.result.model.Try method*), 482
deepcopy() (*robot.result.model.TryBranch method*), 479
deepcopy() (*robot.result.model.While method*), 473
deepcopy() (*robot.result.model.WhileIteration method*), 470
deepcopy() (*robot.running.model.Break method*), 569
deepcopy() (*robot.running.model.Continue method*), 568
deepcopy() (*robot.running.model.For method*), 558
deepcopy() (*robot.running.model.If method*), 563
deepcopy() (*robot.running.model.IfBranch method*), 561
deepcopy() (*robot.running.model.Keyword method*), 556
deepcopy() (*robot.running.model.Return method*), 567
deepcopy() (*robot.running.model.TestCase method*), 570
deepcopy() (*robot.running.model.TestSuite method*), 574
deepcopy() (*robot.running.model.Try method*), 565
deepcopy() (*robot.running.model.TryBranch method*), 564
deepcopy() (*robot.running.model.While method*), 560
default_repr (*robot.running.arguments.argumentspec.ArgInfo attribute*), 530
DEFAULT_TAGS (*robot.parsing.lexer.tokens.END attribute*), 338
DEFAULT_TAGS (*robot.parsing.lexer.tokens.EOS attribute*), 336
DEFAULT_TAGS (*robot.parsing.lexer.tokens.Token attribute*), 334
DefaultLogger (*class in robot.utils.application*), 591
Defaults (*class in robot.running.builder.settings*), 543
DefaultTags (*class in robot.parsing.model.statements*), 355
DefaultValue (*class in robot.running.arguments.argumentmapper*), 529
deiconify() (*robot.libraries.dialogs_py.InputDialog method*), 174
deiconify() (*robot.libraries.dialogs_py.MessageDialog method*), 160
deiconify() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 202
deiconify() (*robot.libraries.dialogs_py.PassFailDialog method*), 216
deiconify() (*robot.libraries.dialogs_py.SelectionDialog method*), 188
delayed_logging (*robot.output.logger.Logger attribute*), 308
deletecommand() (*robot.libraries.dialogs_py.InputDialog method*), 174
deletecommand() (*robot.libraries.dialogs_py.MessageDialog method*), 160
deletecommand() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 202
deletecommand() (*robot.libraries.dialogs_py.PassFailDialog method*), 216
deletecommand() (*robot.libraries.dialogs_py.SelectionDialog method*), 188
deleter() (*robot.utils.misc.classproperty method*), 604
denominator (*robot.reporting.stringcache.StringIndex attribute*), 401
deprecate_tags_starting_with_hyphen() (*in module robot.running.builder.transformers*), 548
deprecated() (*in module robot.result.modeldeprecation*), 499
DeprecatedAttributesMixin (*class in robot.result.modeldeprecation*), 499
deprecation_message (*robot.model.keyword.Keywords attribute*), 261
destroy() (*robot.libraries.dialogs_py.InputDialog method*), 174
destroy() (*robot.libraries.dialogs_py.MessageDialog method*), 160
destroy() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 202
destroy() (*robot.libraries.dialogs_py.PassFailDialog method*), 216
destroy() (*robot.libraries.dialogs_py.SelectionDialog method*), 188
DictDumper (*class in robot.htmldata.jsonwriter*), 68
dictionaries_should_be_equal() (*robot.libraries.Collections.Collections method*), 102
dictionary_should_contain_item() (*robot.libraries.Collections.Collections method*), 102
dictionary_should_contain_key() (*robot.libraries.Collections.Collections method*), 102
dictionary_should_contain_sub_dictionary() (*robot.libraries.Collections.Collections method*), 103

dictionary_should_contain_value() (*robot.libraries.Collections.Collections method*), 103
dictionary_should_not_contain_key() (*robot.libraries.Collections.Collections method*), 103
dictionary_should_not_contain_value() (*robot.libraries.Collections.Collections method*), 103
DictionaryConverter (*class in robot.running.arguments.typeconverters*), 539
DictToKwargs (*class in robot.running.arguments.argumentresolver*), 529
DictVariableTableValue (*class in robot.variables.tablesetter*), 612
directory (*robot.running.model.Import attribute*), 576
directory_should_be_empty() (*robot.libraries.OperatingSystem.OperatingSystem method*), 116
directory_should_exist() (*robot.libraries.OperatingSystem.OperatingSystem method*), 116
directory_should_not_be_empty() (*robot.libraries.OperatingSystem.OperatingSystem method*), 116
directory_should_not_exist() (*robot.libraries.OperatingSystem.OperatingSystem method*), 116
disable_library_import_logging() (*robot.output.logger.Logger method*), 308
disable_message_cache() (*robot.output.logger.Logger method*), 308
discard_suite_scope() (*robot.output.listenermethods.LibraryListenerMethods method*), 307
discard_suite_scope() (*robot.output.listeners.LibraryListeners method*), 307
doc (*robot.libdocpkg.model.LibraryDoc attribute*), 71
doc (*robot.model.keyword.Keyword attribute*), 259
doc (*robot.model.stats.TagStat attribute*), 276
doc (*robot.model.testcase.TestCase attribute*), 283
doc (*robot.model.testsuite.TestSuite attribute*), 285
doc (*robot.result.model.Break attribute*), 488
doc (*robot.result.model.Continue attribute*), 485
doc (*robot.result.model.For attribute*), 467
doc (*robot.result.model.ForIteration attribute*), 465
doc (*robot.result.model.If attribute*), 476
doc (*robot.result.model.IfBranch attribute*), 474
doc (*robot.result.model.Keyword attribute*), 492
doc (*robot.result.model.Return attribute*), 483
doc (*robot.result.model.TestCase attribute*), 494
doc (*robot.result.model.TestSuite attribute*), 496
doc (*robot.result.model.Try attribute*), 481
doc (*robot.result.model.TryBranch attribute*), 478
doc (*robot.result.model.While attribute*), 472
doc (*robot.result.model.WhileIteration attribute*), 469
doc (*robot.running.arguments.customconverters.ConverterInfo attribute*), 531
doc (*robot.running.arguments.typeconverters.BooleanConverter attribute*), 533
doc (*robot.running.arguments.typeconverters.ByteArrayConverter attribute*), 535
doc (*robot.running.arguments.typeconverters.BytesConverter attribute*), 535
doc (*robot.running.arguments.typeconverters.CombinedConverter attribute*), 540
doc (*robot.running.arguments.typeconverters.CustomConverter attribute*), 541
doc (*robot.running.arguments.typeconverters.DateConverter attribute*), 536
doc (*robot.running.arguments.typeconverters.DateTimeConverter attribute*), 536
doc (*robot.running.arguments.typeconverters.DecimalConverter attribute*), 534
doc (*robot.running.arguments.typeconverters.DictionaryConverter attribute*), 539
doc (*robot.running.arguments.typeconverters.EnumConverter attribute*), 532
doc (*robot.running.arguments.typeconverters.FloatConverter attribute*), 534
doc (*robot.running.arguments.typeconverters.FrozenSetConverter attribute*), 540
doc (*robot.running.arguments.typeconverters.IntegerConverter attribute*), 533
doc (*robot.running.arguments.typeconverters.ListConverter attribute*), 538
doc (*robot.running.arguments.typeconverters.NoneConverter attribute*), 537
doc (*robot.running.arguments.typeconverters.PathConverter attribute*), 537
doc (*robot.running.arguments.typeconverters.SetConverter attribute*), 539
doc (*robot.running.arguments.typeconverters.StringConverter attribute*), 532
doc (*robot.running.arguments.typeconverters.TimeDeltaConverter attribute*), 536
doc (*robot.running.arguments.typeconverters.TupleConverter attribute*), 538
doc (*robot.running.arguments.typeconverters.TypeConverter attribute*), 532
doc (*robot.running.arguments.typeconverters.TypedDictConverter attribute*), 539
doc (*robot.running.model.Keyword attribute*), 556
doc (*robot.running.model.TestCase attribute*), 571

doc (*robot.running.model.TestSuite attribute*), 574
doc (*robot.running.usererrorhandler.UserErrorHandler attribute*), 589
doc_format (*robot.libdocpkg.model.LibraryDoc attribute*), 71
DocFormatter (*class in robot.libdocpkg.htmlutils*), 70
DocHandler (*class in robot.result.xmlelementhandlers*), 523
DocToHtml (*class in robot.libdocpkg.htmlutils*), 70
Documentation (*class in robot.parsing.model.statements*), 353
DOCUMENTATION (*robot.parsing.lexer.tokens.END attribute*), 338
DOCUMENTATION (*robot.parsing.lexer.tokens.EOS attribute*), 336
DOCUMENTATION (*robot.parsing.lexer.tokens.Token attribute*), 334
documentation_setting (*robot.conf.languages.Bg attribute*), 59
documentation_setting (*robot.conf.languages.Bs attribute*), 38
documentation_setting (*robot.conf.languages.Cs attribute*), 36
documentation_setting (*robot.conf.languages.De attribute*), 42
documentation_setting (*robot.conf.languages.En attribute*), 34
documentation_setting (*robot.conf.languages.Es attribute*), 51
documentation_setting (*robot.conf.languages.Fi attribute*), 40
documentation_setting (*robot.conf.languages.Fr attribute*), 41
documentation_setting (*robot.conf.languages.Hi attribute*), 63
documentation_setting (*robot.conf.languages.It attribute*), 62
documentation_setting (*robot.conf.languages.Language attribute*), 33
documentation_setting (*robot.conf.languages.Nl attribute*), 37
documentation_setting (*robot.conf.languages.Pl attribute*), 48
documentation_setting (*robot.conf.languages.Pt attribute*), 45
documentation_setting (*robot.conf.languages.PtBr attribute*), 44
documentation_setting (*robot.conf.languages.Ro attribute*), 61
documentation_setting (*robot.conf.languages.Ru attribute*), 52
documentation_setting (*robot.conf.languages.Sv attribute*), 58
documentation_setting (*robot.conf.languages.Th attribute*), 47
documentation_setting (*robot.conf.languages.Tr attribute*), 56
documentation_setting (*robot.conf.languages.Uk attribute*), 49
documentation_setting (*robot.conf.languages.ZhCn attribute*), 54
documentation_setting (*robot.conf.languages.ZhTw attribute*), 55
DocumentationBuilder (*class in robot.libdocpkg.builder*), 69
DocumentationOrMetadata (*class in robot.parsing.model.statements*), 346
dont_continue (*robot.errors.BreakLoop attribute*), 617
dont_continue (*robot.errors.ContinueLoop attribute*), 617
dont_continue (*robot.errors.ExecutionFailed attribute*), 615
dont_continue (*robot.errors.ExecutionFailures attribute*), 616
dont_continue (*robot.errors.ExecutionPassed attribute*), 616
dont_continue (*robot.errors.ExecutionStatus attribute*), 614
dont_continue (*robot.errors.HandlerExecutionFailed attribute*), 615
dont_continue (*robot.errors.PassExecution attribute*), 617
dont_continue (*robot.errors.ReturnFromKeyword attribute*), 618
dont_continue (*robot.errors.UserKeywordExecutionFailed attribute*), 616
DosHighlighter (*class in robot.output.console.highlighting*), 303
DotDict (*class in robot.utils.dotdict*), 596
DottedImporter (*class in robot.utils.importer*), 602
DottedOutput (*class in robot.output.console.dotted*), 298
dry_run (*robot.conf.settings.RobotSettings attribute*), 65
dry_run() (*robot.running.librarykeywordrunner.EmbeddedArgumentsRu method*), 552
dry_run() (*robot.running.librarykeywordrunner.LibraryKeywordRunner method*), 552
dry_run() (*robot.running.librarykeywordrunner.RunKeywordRunner method*), 552
dry_run() (*robot.running.usererrorhandler.UserErrorHandler method*), 589
dry_run() (*robot.running.userkeywordrunner.EmbeddedArgumentsRunn method*), 590
dry_run() (*robot.running.userkeywordrunner.UserKeywordRunner method*), 590

dump() (*robot.htmldata.jsonwriter.DictDumper method*), 68
dump() (*robot.htmldata.jsonwriter.IntegerDumper method*), 68
dump() (*robot.htmldata.jsonwriter.JsonDumper method*), 68
dump() (*robot.htmldata.jsonwriter.MappingDumper method*), 68
dump() (*robot.htmldata.jsonwriter.NoneDumper method*), 68
dump() (*robot.htmldata.jsonwriter.StringDumper method*), 68
dump() (*robot.htmldata.jsonwriter.TupleListDumper method*), 68
dump() (*robot.reporting.stringcache.StringCache method*), 402
DurationLimit (*class in robot.running.bodyrunner*), 550
DynamicArgumentParser (*class in robot.running.arguments.argumentparser*), 529
DynamicHandler() (*in module robot.running.handlers*), 551

E

earlier_failures (*robot.errors.BreakLoop attribute*), 617
earlier_failures (*robot.errors.ContinueLoop attribute*), 617
earlier_failures (*robot.errors.ExecutionPassed attribute*), 616
earlier_failures (*robot.errors.PassExecution attribute*), 617
earlier_failures (*robot.errors.ReturnFromKeyword attribute*), 618
elapsed (*robot.model.stats.Stat attribute*), 275
elapsed (*robot.model.stats.SuiteStat attribute*), 275
elapsedtime (*robot.result.model.Break attribute*), 489
elapsedtime (*robot.result.model.Continue attribute*), 487
elapsedtime (*robot.result.model.For attribute*), 468
elapsedtime (*robot.result.model.ForIteration attribute*), 466
elapsedtime (*robot.result.model.If attribute*), 477
elapsedtime (*robot.result.model.IfBranch attribute*), 475
elapsedtime (*robot.result.model.Keyword attribute*), 492
elapsedtime (*robot.result.model.Return attribute*), 484
elapsedtime (*robot.result.model.StatusMixin attribute*), 464
elapsedtime (*robot.result.model.TestCase attribute*), 494
elapsedtime (*robot.result.model.TestSuite attribute*), 499
elapsedtime (*robot.result.model.Try attribute*), 482
elapsedtime (*robot.result.model.TryBranch attribute*), 480
elapsedtime (*robot.result.model.While attribute*), 473
elapsedtime (*robot.result.model.WhileIteration attribute*), 471
element() (*robot.utils.markupwriters.HtmlWriter method*), 602
element() (*robot.utils.markupwriters.NullMarkupWriter method*), 603
element() (*robot.utils.markupwriters.XmlWriter method*), 603
element_attribute_should_be() (*robot.libraries.XML.XML method*), 153
element_attribute_should_match() (*robot.libraries.XML.XML method*), 153
element_handlers (*robot.result.xmlelementhandlers.ArgumentHandle attribute*), 526
element_handlers (*robot.result.xmlelementhandlers.ArgumentsHandl attribute*), 525
element_handlers (*robot.result.xmlelementhandlers.AssignHandler attribute*), 525
element_handlers (*robot.result.xmlelementhandlers.BranchHandler attribute*), 521
element_handlers (*robot.result.xmlelementhandlers.BreakHandler attribute*), 522
element_handlers (*robot.result.xmlelementhandlers.ContinueHandler attribute*), 522
element_handlers (*robot.result.xmlelementhandlers.DocHandler attribute*), 523
element_handlers (*robot.result.xmlelementhandlers.ElementHandler attribute*), 519
element_handlers (*robot.result.xmlelementhandlers.ErrorMessageHa attribute*), 526
element_handlers (*robot.result.xmlelementhandlers.ErrorsHandler attribute*), 526
element_handlers (*robot.result.xmlelementhandlers.ForHandler attribute*), 520
element_handlers (*robot.result.xmlelementhandlers.IfHandler attribute*), 521
element_handlers (*robot.result.xmlelementhandlers.IterationHandler attribute*), 521
element_handlers (*robot.result.xmlelementhandlers.KeywordHandler attribute*), 520
element_handlers (*robot.result.xmlelementhandlers.MessageHandler attribute*), 523
element_handlers (*robot.result.xmlelementhandlers.MetadataHandle attribute*), 523
element_handlers (*robot.result.xmlelementhandlers.MetadataItemHa attribute*), 524
element_handlers (*robot.result.xmlelementhandlers.MetaHandler attribute*), 524

element_handlers (*robot.result.xmlelementhandlers.ParseHandler attribute*), 522
element_handlers (*robot.result.xmlelementhandlers.ReturnHandler attribute*), 522
element_handlers (*robot.result.xmlelementhandlers.RobotHandler attribute*), 519
element_handlers (*robot.result.xmlelementhandlers.RootHandler attribute*), 519
element_handlers (*robot.result.xmlelementhandlers.StatisticsHandler attribute*), 527
element_handlers (*robot.result.xmlelementhandlers.StatusHandler attribute*), 523
element_handlers (*robot.result.xmlelementhandlers.SuiteHandler attribute*), 520
element_handlers (*robot.result.xmlelementhandlers.TagHandler attribute*), 524
element_handlers (*robot.result.xmlelementhandlers.TagsHandler attribute*), 524
element_handlers (*robot.result.xmlelementhandlers.TestHandler attribute*), 520
element_handlers (*robot.result.xmlelementhandlers.TimeHandler attribute*), 525
element_handlers (*robot.result.xmlelementhandlers.TryHandler attribute*), 521
element_handlers (*robot.result.xmlelementhandlers.ValueHandler attribute*), 526
element_handlers (*robot.result.xmlelementhandlers.VarHandler attribute*), 525
element_handlers (*robot.result.xmlelementhandlers.WhileHandler attribute*), 520
element_should_exist() (*robot.libraries.XML.XML method*), 151
element_should_not_exist() (*robot.libraries.XML.XML method*), 152
element_should_not_have_attribute() (*robot.libraries.XML.XML method*), 153
element_text_should_be() (*robot.libraries.XML.XML method*), 152
element_text_should_match() (*robot.libraries.XML.XML method*), 152
element_to_string() (*robot.libraries.XML.XML method*), 156
ElementComparator (*class in robot.libraries.XML*), 157
ElementFinder (*class in robot.libraries.XML*), 157
ElementHandler (*class in robot.result.xmlelementhandlers*), 519
elements_should_be_equal() (*robot.libraries.XML.XML method*), 153
elements_should_match() (*robot.libraries.XML.XML method*), 154
ELSE (*robot.model.body.BodyItem attribute*), 228
ELSE (*robot.model.control.Break attribute*), 249
ELSE (*robot.model.control.Continue attribute*), 248
ELSE (*robot.model.control.For attribute*), 239
ELSE (*robot.model.control.If attribute*), 243
ELSE (*robot.model.control.IfBranch attribute*), 241
ELSE (*robot.model.control.Return attribute*), 247
ELSE (*robot.model.control.Try attribute*), 245
ELSE (*robot.model.control.TryBranch attribute*), 244
ELSE (*robot.model.control.While attribute*), 240
ELSE (*robot.model.keyword.Keyword attribute*), 260
ELSE (*robot.model.message.Message attribute*), 263
ELSE (*robot.output.loggerhelper.Message attribute*), 309
ELSE (*robot.parsing.lexer.tokens.END attribute*), 338
ELSE (*robot.parsing.lexer.tokens.EOS attribute*), 336
ELSE (*robot.parsing.lexer.tokens.Token attribute*), 335
ELSE (*robot.result.model.Break attribute*), 488
ELSE (*robot.result.model.Continue attribute*), 486
ELSE (*robot.result.model.For attribute*), 467
ELSE (*robot.result.model.ForIteration attribute*), 465
ELSE (*robot.result.model.If attribute*), 476
ELSE (*robot.result.model.IfBranch attribute*), 474
ELSE (*robot.result.model.Keyword attribute*), 491
ELSE (*robot.result.model.Message attribute*), 463
ELSE (*robot.result.model.Return attribute*), 483
ELSE (*robot.result.model.Try attribute*), 481
ELSE (*robot.result.model.TryBranch attribute*), 478
ELSE (*robot.result.model.While attribute*), 472
ELSE (*robot.result.model.WhileIteration attribute*), 470
ELSE (*robot.running.model.Break attribute*), 569
ELSE (*robot.running.model.Continue attribute*), 567
ELSE (*robot.running.model.For attribute*), 558
ELSE (*robot.running.model.If attribute*), 562
ELSE (*robot.running.model.IfBranch attribute*), 560
ELSE (*robot.running.model.Keyword attribute*), 555
ELSE (*robot.running.model.Return attribute*), 566
ELSE (*robot.running.model.Try attribute*), 565
ELSE (*robot.running.model.TryBranch attribute*), 563
ELSE (*robot.running.model.While attribute*), 559
else_branch (*robot.model.control.Try attribute*), 245
else_branch (*robot.result.model.Try attribute*), 482
else_branch (*robot.running.model.Try attribute*), 565
ELSE_IF (*robot.model.body.BodyItem attribute*), 228
ELSE_IF (*robot.model.control.Break attribute*), 249
ELSE_IF (*robot.model.control.Continue attribute*), 248
ELSE_IF (*robot.model.control.For attribute*), 239
ELSE_IF (*robot.model.control.If attribute*), 243
ELSE_IF (*robot.model.control.IfBranch attribute*), 241
ELSE_IF (*robot.model.control.Return attribute*), 247
ELSE_IF (*robot.model.control.Try attribute*), 245
ELSE_IF (*robot.model.control.TryBranch attribute*), 244
ELSE_IF (*robot.model.control.While attribute*), 240
ELSE_IF (*robot.model.keyword.Keyword attribute*), 260
ELSE_IF (*robot.model.message.Message attribute*), 263
ELSE_IF (*robot.output.loggerhelper.Message attribute*), 309

ELSE_IF (*robot.parsing.lexer.tokens.END attribute*), 338
ELSE_IF (*robot.parsing.lexer.tokens.EOS attribute*), 336
ELSE_IF (*robot.parsing.lexer.tokens.Token attribute*), 335
ELSE_IF (*robot.result.model.Break attribute*), 488
ELSE_IF (*robot.result.model.Continue attribute*), 486
ELSE_IF (*robot.result.model.For attribute*), 467
ELSE_IF (*robot.result.model.ForIteration attribute*), 465
ELSE_IF (*robot.result.model.If attribute*), 476
ELSE_IF (*robot.result.model.IfBranch attribute*), 474
ELSE_IF (*robot.result.model.Keyword attribute*), 491
ELSE_IF (*robot.result.model.Message attribute*), 463
ELSE_IF (*robot.result.model.Return attribute*), 483
ELSE_IF (*robot.result.model.Try attribute*), 481
ELSE_IF (*robot.result.model.TryBranch attribute*), 479
ELSE_IF (*robot.result.model.While attribute*), 472
ELSE_IF (*robot.result.model.WhileIteration attribute*), 470
ELSE_IF (*robot.running.model.Break attribute*), 569
ELSE_IF (*robot.running.model.Continue attribute*), 567
ELSE_IF (*robot.running.model.For attribute*), 558
ELSE_IF (*robot.running.model.If attribute*), 562
ELSE_IF (*robot.running.model.IfBranch attribute*), 560
ELSE_IF (*robot.running.model.Keyword attribute*), 555
ELSE_IF (*robot.running.model.Return attribute*), 566
ELSE_IF (*robot.running.model.Try attribute*), 565
ELSE_IF (*robot.running.model.TryBranch attribute*), 563
ELSE_IF (*robot.running.model.While attribute*), 559
ElseHeader (*class in robot.parsing.model.statements*), 376
ElseHeaderLexer (*class in robot.parsing.lexer.statementlexers*), 331
ElseIfHeader (*class in robot.parsing.model.statements*), 375
ElseIfHeaderLexer (*class in robot.parsing.lexer.statementlexers*), 331
EmbeddedArgumentParser (*class in robot.running.arguments.embedded*), 531
EmbeddedArguments (*class in robot.running.arguments.embedded*), 531
EmbeddedArgumentsHandler (*class in robot.running.handlers*), 551
EmbeddedArgumentsHandler (*class in robot.running.userkeyword*), 590
EmbeddedArgumentsRunner (*class in robot.running.librarykeywordrunner*), 552
EmbeddedArgumentsRunner (*class in robot.running.userkeywordrunner*), 590
emit() (*robot.output.pyloggingconf.RobotHandler method*), 312
empty (*robot.variables.finders.EmptyFinder attribute*), 608
empty_cache() (*robot.utils.connectioncache.ConnectionCache method*), 596
empty_directory() (*robot.libraries.OperatingSystem.OperatingSystem method*), 117
EmptyFinder (*class in robot.variables.finders*), 608
EmptyLine (*class in robot.parsing.model.statements*), 387
EmptySuiteRemover (*class in robot.model.filter*), 250
En (*class in robot.conf.languages*), 34
enable_library_import_logging() (*robot.output.logger.Logger method*), 308
encode_string_to_bytes() (*robot.libraries.String.String method*), 132
encode_threshold (*robot.libraries.Remote.TimeoutHTTPSTransport attribute*), 129
encode_threshold (*robot.libraries.Remote.TimeoutHTTPTransport attribute*), 129
END (*class in robot.parsing.lexer.tokens*), 338
End (*class in robot.parsing.model.statements*), 380
END (*robot.parsing.lexer.tokens.END attribute*), 338
END (*robot.parsing.lexer.tokens.EOS attribute*), 336
END (*robot.parsing.lexer.tokens.Token attribute*), 335
end() (*robot.result.xmlelementhandlers.ArgumentHandler method*), 525
end() (*robot.result.xmlelementhandlers.ArgumentsHandler method*), 525
end() (*robot.result.xmlelementhandlers.AssignHandler method*), 525
end() (*robot.result.xmlelementhandlers.BranchHandler method*), 521
end() (*robot.result.xmlelementhandlers.BreakHandler method*), 522
end() (*robot.result.xmlelementhandlers.ContinueHandler method*), 522
end() (*robot.result.xmlelementhandlers.DocHandler method*), 523
end() (*robot.result.xmlelementhandlers.ElementHandler method*), 519
end() (*robot.result.xmlelementhandlers.ErrorMessageHandler method*), 526
end() (*robot.result.xmlelementhandlers.ErrorsHandler method*), 526
end() (*robot.result.xmlelementhandlers.ForHandler method*), 520
end() (*robot.result.xmlelementhandlers.IfHandler method*), 521
end() (*robot.result.xmlelementhandlers.IterationHandler method*), 521
end() (*robot.result.xmlelementhandlers.KeywordHandler method*), 520

end() (*robot.result.xmlelementhandlers.MessageHandler method*), 523
end() (*robot.result.xmlelementhandlers.MetadataHandler method*), 523
end() (*robot.result.xmlelementhandlers.MetadataItemHandler method*), 524
end() (*robot.result.xmlelementhandlers.MetaHandler method*), 524
end() (*robot.result.xmlelementhandlers.PatternHandler method*), 522
end() (*robot.result.xmlelementhandlers.ReturnHandler method*), 522
end() (*robot.result.xmlelementhandlers.RobotHandler method*), 519
end() (*robot.result.xmlelementhandlers.RootHandler method*), 519
end() (*robot.result.xmlelementhandlers.StatisticsHandler method*), 527
end() (*robot.result.xmlelementhandlers.StatusHandler method*), 523
end() (*robot.result.xmlelementhandlers.SuiteHandler method*), 520
end() (*robot.result.xmlelementhandlers.TagHandler method*), 524
end() (*robot.result.xmlelementhandlers.TagsHandler method*), 524
end() (*robot.result.xmlelementhandlers.TestHandler method*), 520
end() (*robot.result.xmlelementhandlers.TimeoutHandler method*), 525
end() (*robot.result.xmlelementhandlers.TryHandler method*), 521
end() (*robot.result.xmlelementhandlers.ValueHandler method*), 526
end() (*robot.result.xmlelementhandlers.VarHandler method*), 525
end() (*robot.result.xmlelementhandlers.WhileHandler method*), 520
end() (*robot.result.xmlelementhandlers.XmlElementHandler method*), 519
end() (*robot.utils.htmlformatters.HeaderFormatter method*), 599
end() (*robot.utils.htmlformatters.ListFormatter method*), 600
end() (*robot.utils.htmlformatters.ParagraphFormatter method*), 600
end() (*robot.utils.htmlformatters.PreformattedFormatter method*), 600
end() (*robot.utils.htmlformatters.RulerFormatter method*), 599
end() (*robot.utils.htmlformatters.TableFormatter method*), 600
end() (*robot.utils.markupwriters.HtmlWriter method*), 602
end() (*robot.utils.markupwriters.NullMarkupWriter method*), 603
end() (*robot.utils.markupwriters.XmlWriter method*), 603
end_block() (*robot.parsing.model.blocks.ValidationContext method*), 345
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*), 234
end_body_item() (*robot.model.filter.EmptySuiteRemover method*), 250
end_body_item() (*robot.model.filter.Filter method*), 254
end_body_item() (*robot.model.modifier.ModelModifier method*), 265
end_body_item() (*robot.model.statistics.StatisticsBuilder method*), 270
end_body_item() (*robot.model.tagsetter.TagSetter method*), 278
end_body_item() (*robot.model.totalstatistics.TotalStatisticsBuilder method*), 289
end_body_item() (*robot.model.visitor.SuiteVisitor method*), 298
end_body_item() (*robot.output.console.dotted.StatusReporter method*), 299
end_body_item() (*robot.output.xmllogger.XmlLogger method*), 316
end_body_item() (*robot.reporting.outputwriter.OutputWriter method*), 396
end_body_item() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 403
end_body_item() (*robot.result.configurer.SuiteConfigurer method*), 408
end_body_item() (*robot.result.keywordremover.AllKeywordsRemover method*), 416
end_body_item() (*robot.result.keywordremover.ByNameKeywordRemo method*), 424
end_body_item() (*robot.result.keywordremover.ByTagKeywordRemove method*), 428
end_body_item() (*robot.result.keywordremover.ForLoopItemsRemover method*), 432
end_body_item() (*robot.result.keywordremover.PassedKeywordRemov method*), 420
end_body_item() (*robot.result.keywordremover.WaitUntilKeywordSucc method*), 440
end_body_item() (*robot.result.keywordremover.WarningAndErrorFind method*), 445
end_body_item() (*robot.result.keywordremover.WhileLoopItemsRemov method*), 436
end_body_item() (*robot.result.merger.Merger method*), 449

end_body_item() (*robot.result.messagefilter.MessageFilter method*), 453
end_body_item() (*robot.result.resultbuilder.RemoveKeywords method*), 501
end_body_item() (*robot.result.suiteteardownfailed.SuiteTeardownFailed method*), 509
end_body_item() (*robot.result.suiteteardownfailed.SuiteTeardownFailureHandler method*), 505
end_body_item() (*robot.result.visitor.ResultVisitor method*), 514
end_body_item() (*robot.running.randomizer.Randomizer method*), 578
end_body_item() (*robot.running.suiterunner.SuiteRunner method*), 585
end_break() (*robot.conf.gatherfailed.GatherFailedSuites method*), 28
end_break() (*robot.conf.gatherfailed.GatherFailedTests method*), 23
end_break() (*robot.model.configurer.SuiteConfigurer method*), 234
end_break() (*robot.model.filter.EmptySuiteRemover method*), 250
end_break() (*robot.model.filter.Filter method*), 255
end_break() (*robot.model.modifier.ModelModifier method*), 266
end_break() (*robot.model.statistics.StatisticsBuilder method*), 271
end_break() (*robot.model.tagsetter.TagSetter method*), 278
end_break() (*robot.model.totalstatistics.TotalStatisticsBuilder method*), 289
end_break() (*robot.model.visitor.SuiteVisitor method*), 297
end_break() (*robot.output.console.dotted.StatusReporter method*), 299
end_break() (*robot.output.xmllogger.XmlLogger method*), 315
end_break() (*robot.reporting.outputwriter.OutputWriter method*), 396
end_break() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 403
end_break() (*robot.result.configurer.SuiteConfigurer method*), 408
end_break() (*robot.result.keywordremover.AllKeywordsRemover method*), 416
end_break() (*robot.result.keywordremover.ByNameKeywordRemover method*), 424
end_break() (*robot.result.keywordremover.ByTagKeywordRemover method*), 428
end_break() (*robot.result.keywordremover.ForLoopItemsRemover method*), 432
end_break() (*robot.result.keywordremover.PassedKeywordRemover method*), 420
end_break() (*robot.result.keywordremover.WaitUntilKeywordSucceedsRemover method*), 441
end_break() (*robot.result.keywordremover.WarningAndErrorFinder method*), 445
end_break() (*robot.result.keywordremover.WhileLoopItemsRemover method*), 436
end_break() (*robot.result.merger.Merger method*), 449
end_break() (*robot.result.messagefilter.MessageFilter method*), 454
end_break() (*robot.result.resultbuilder.RemoveKeywords method*), 501
end_break() (*robot.result.suiteteardownfailed.SuiteTeardownFailed method*), 509
end_break() (*robot.result.suiteteardownfailed.SuiteTeardownFailureHa method*), 505
end_break() (*robot.result.visitor.ResultVisitor method*), 514
end_break() (*robot.running.randomizer.Randomizer method*), 579
end_break() (*robot.running.suiterunner.SuiteRunner method*), 585
end_col_offset (*robot.parsing.lexer.tokens.END attribute*), 340
end_col_offset (*robot.parsing.lexer.tokens.EOS attribute*), 338
end_col_offset (*robot.parsing.lexer.tokens.Token attribute*), 335
end_col_offset (*robot.parsing.model.blocks.Block attribute*), 340
end_col_offset (*robot.parsing.model.blocks.CommentSection attribute*), 342
end_col_offset (*robot.parsing.model.blocks.File attribute*), 341
end_col_offset (*robot.parsing.model.blocks.For attribute*), 344
end_col_offset (*robot.parsing.model.blocks.HeaderAndBody attribute*), 341
end_col_offset (*robot.parsing.model.blocks.If attribute*), 343
end_col_offset (*robot.parsing.model.blocks.Keyword attribute*), 343
end_col_offset (*robot.parsing.model.blocks.KeywordSection attribute*), 342
end_col_offset (*robot.parsing.model.blocks.Section attribute*), 341
end_col_offset (*robot.parsing.model.blocks.SettingSection attribute*), 341
end_col_offset (*robot.parsing.model.blocks.TestCase attribute*), 343
end_col_offset (*robot.parsing.model.blocks.TestCaseSection attribute*), 342
end_col_offset (*robot.parsing.model.blocks.Try attribute*), 344
end_col_offset (*robot.parsing.model.blocks.VariableSection

attribute), 342
end_col_offset (*robot.parsing.model.blocks.While attribute*), 344
end_col_offset (*robot.parsing.model.statements.Arguments attribute*), 369
end_col_offset (*robot.parsing.model.statements.Break attribute*), 384
end_col_offset (*robot.parsing.model.statements.Comment attribute*), 385
end_col_offset (*robot.parsing.model.statements.Config attribute*), 386
end_col_offset (*robot.parsing.model.statements.Continue attribute*), 383
end_col_offset (*robot.parsing.model.statements.DefaultTags attribute*), 356
end_col_offset (*robot.parsing.model.statements.Documentation attribute*), 353
end_col_offset (*robot.parsing.model.statements.DocumentationOrMetadata attribute*), 346
end_col_offset (*robot.parsing.model.statements.ElseHeader attribute*), 376
end_col_offset (*robot.parsing.model.statements.ElseIfHeader attribute*), 375
end_col_offset (*robot.parsing.model.statements.EmptyLine attribute*), 388
end_col_offset (*robot.parsing.model.statements.End attribute*), 380
end_col_offset (*robot.parsing.model.statements.Error attribute*), 387
end_col_offset (*robot.parsing.model.statements.ExceptHeader attribute*), 379
end_col_offset (*robot.parsing.model.statements.FinallyHeader attribute*), 379
end_col_offset (*robot.parsing.model.statements.Fixture attribute*), 349
end_col_offset (*robot.parsing.model.statements.ForceTags attribute*), 355
end_col_offset (*robot.parsing.model.statements.ForHeader attribute*), 372
end_col_offset (*robot.parsing.model.statements.IfElseHeader attribute*), 373
end_col_offset (*robot.parsing.model.statements.IfHeader attribute*), 374
end_col_offset (*robot.parsing.model.statements.InlineIfHeader attribute*), 374
end_col_offset (*robot.parsing.model.statements.KeywordCall attribute*), 370
end_col_offset (*robot.parsing.model.statements.KeywordName attribute*), 364
end_col_offset (*robot.parsing.model.statements.KeywordTags attribute*), 356
end_col_offset (*robot.parsing.model.statements.LibraryImport attribute*), 351
end_col_offset (*robot.parsing.model.statements.LoopControl attribute*), 383
end_col_offset (*robot.parsing.model.statements.Metadata attribute*), 354
end_col_offset (*robot.parsing.model.statements.MultiValue attribute*), 348
end_col_offset (*robot.parsing.model.statements.NoArgumentHeader attribute*), 377
end_col_offset (*robot.parsing.model.statements.ResourceImport attribute*), 351
end_col_offset (*robot.parsing.model.statements.Return attribute*), 369
end_col_offset (*robot.parsing.model.statements.ReturnStatement attribute*), 382
end_col_offset (*robot.parsing.model.statements.SectionHeader attribute*), 350
end_col_offset (*robot.parsing.model.statements.Setup attribute*), 365
end_col_offset (*robot.parsing.model.statements.SingleValue attribute*), 347
end_col_offset (*robot.parsing.model.statements.Statement attribute*), 346
end_col_offset (*robot.parsing.model.statements.SuiteSetup attribute*), 357
end_col_offset (*robot.parsing.model.statements.SuiteTeardown attribute*), 358
end_col_offset (*robot.parsing.model.statements.Tags attribute*), 366
end_col_offset (*robot.parsing.model.statements.Teardown attribute*), 365
end_col_offset (*robot.parsing.model.statements.Template attribute*), 367
end_col_offset (*robot.parsing.model.statements.TemplateArguments attribute*), 371
end_col_offset (*robot.parsing.model.statements.TestCaseName attribute*), 363
end_col_offset (*robot.parsing.model.statements.TestSetup attribute*), 359
end_col_offset (*robot.parsing.model.statements.TestTeardown attribute*), 360
end_col_offset (*robot.parsing.model.statements.TestTemplate attribute*), 361
end_col_offset (*robot.parsing.model.statements.TestTimeout attribute*), 361
end_col_offset (*robot.parsing.model.statements.Timeout attribute*), 368
end_col_offset (*robot.parsing.model.statements.TryHeader attribute*), 378
end_col_offset (*robot.parsing.model.statements.Variable attribute*), 362
end_col_offset (*robot.parsing.model.statements.VariablesImport attribute*), 352
end_col_offset (*robot.parsing.model.statements.WhileHeader attribute*), 381
end_continue() (*robot.conf.gatherfailed.GatherFailedSuites*

method), 28
end_continue() (*robot.conf.gatherfailed.GatherFailedTests method*), 23
end_continue() (*robot.model.configurer.SuiteConfigurer method*), 234
end_continue() (*robot.model.filter.EmptySuiteRemover method*), 250
end_continue() (*robot.model.filter.Filter method*), 255
end_continue() (*robot.model.modifier.ModelModifier method*), 266
end_continue() (*robot.model.statistics.StatisticsBuilder method*), 271
end_continue() (*robot.model.tagsetter.TagSetter method*), 278
end_continue() (*robot.model.totalstatistics.TotalStatisticsBuilder method*), 289
end_continue() (*robot.model.visitor.SuiteVisitor method*), 297
end_continue() (*robot.output.console.dotted.StatusReporter method*), 299
end_continue() (*robot.output.xmllogger.XmlLogger method*), 315
end_continue() (*robot.reporting.outputwriter.OutputWriter method*), 396
end_continue() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 403
end_continue() (*robot.result.configurer.SuiteConfigurer method*), 408
end_continue() (*robot.result.keywordremover.AllKeywordsRemover method*), 416
end_continue() (*robot.result.keywordremover.ByNameKeywordRemover method*), 424
end_continue() (*robot.result.keywordremover.ByTagKeywordRemover method*), 428
end_continue() (*robot.result.keywordremover.ForLoopItemsRemover method*), 432
end_continue() (*robot.result.keywordremover.PassedKeywordRemover method*), 420
end_continue() (*robot.result.keywordremover.WaitUntilKeywordSucceedsRemover method*), 441
end_continue() (*robot.result.keywordremover.WarningAndErrorFinder method*), 445
end_continue() (*robot.result.keywordremover.WhileLoopItemsRemover method*), 436
end_continue() (*robot.result.merger.Merger method*), 449
end_continue() (*robot.result.messagefilter.MessageFilter method*), 454
end_continue() (*robot.result.resultbuilder.RemoveKeywords method*), 501
end_continue() (*robot.result.suiteteardownfailed.SuiteTeardownFailed method*), 510
end_continue() (*robot.result.suiteteardownfailed.SuiteTeardownFailureHandler method*), 505
end_continue() (*robot.result.visitor.ResultVisitor method*), 514
end_continue() (*robot.running.randomizer.Randomizer method*), 579
end_continue() (*robot.running.suiterunner.SuiteRunner method*), 585
end_directory() (*robot.parsing.suitestructure.SuiteStructureVisitor method*), 393
end_directory() (*robot.running.builder.builders.SuiteStructureParser method*), 542
end_errors() (*robot.output.xmllogger.XmlLogger method*), 316
end_errors() (*robot.reporting.outputwriter.OutputWriter method*), 396
end_errors() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 403
end_errors() (*robot.result.visitor.ResultVisitor method*), 514
end_for() (*robot.conf.gatherfailed.GatherFailedSuites method*), 28
end_for() (*robot.conf.gatherfailed.GatherFailedTests method*), 23
end_for() (*robot.model.configurer.SuiteConfigurer method*), 234
end_for() (*robot.model.filter.EmptySuiteRemover method*), 251
end_for() (*robot.model.filter.Filter method*), 255
end_for() (*robot.model.modifier.ModelModifier method*), 266
end_for() (*robot.model.statistics.StatisticsBuilder method*), 271
end_for() (*robot.model.tagsetter.TagSetter method*), 278
end_for() (*robot.model.totalstatistics.TotalStatisticsBuilder method*), 289
end_for() (*robot.model.visitor.SuiteVisitor method*), 295
end_for() (*robot.output.console.dotted.StatusReporter method*), 299
end_for() (*robot.output.xmllogger.XmlLogger method*), 314
end_for() (*robot.reporting.outputwriter.OutputWriter method*), 396
end_for() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 403
end_for() (*robot.result.configurer.SuiteConfigurer method*), 409
end_for() (*robot.result.keywordremover.AllKeywordsRemover method*), 416
end_for() (*robot.result.keywordremover.ByNameKeywordRemover method*), 424
end_for() (*robot.result.keywordremover.ByTagKeywordRemover method*), 428

end_for() (*robot.result.keywordremover.ForLoopItemsRemover method*), 432
end_for() (*robot.result.keywordremover.PassedKeywordRemover method*), 420
end_for() (*robot.result.keywordremover.WaitUntilKeywordSucceedsRemover method*), 441
end_for() (*robot.result.keywordremover.WarningAndErrorFinder method*), 445
end_for() (*robot.result.keywordremover.WhileLoopItemsRemover method*), 437
end_for() (*robot.result.merger.Merger method*), 449
end_for() (*robot.result.messagefilter.MessageFilter method*), 454
end_for() (*robot.result.resultbuilder.RemoveKeywords method*), 501
end_for() (*robot.result.suiteteardownfailed.SuiteTeardownFailed method*), 510
end_for() (*robot.result.suiteteardownfailed.SuiteTeardownFailureHandler method*), 505
end_for() (*robot.result.visitor.ResultVisitor method*), 515
end_for() (*robot.running.randomizer.Randomizer method*), 579
end_for() (*robot.running.suiterunner.SuiteRunner method*), 585
end_for_iteration() (*robot.conf.gatherfailed.GatherFailedSuites method*), 28
end_for_iteration() (*robot.conf.gatherfailed.GatherFailedTests method*), 24
end_for_iteration() (*robot.model.configurer.SuiteConfigurer method*), 234
end_for_iteration() (*robot.model.filter.EmptySuiteRemover method*), 251
end_for_iteration() (*robot.model.filter.Filter method*), 255
end_for_iteration() (*robot.model.modifier.ModelModifier method*), 266
end_for_iteration() (*robot.model.statistics.StatisticsBuilder method*), 271
end_for_iteration() (*robot.model.tagsetter.TagSetter method*), 278
end_for_iteration() (*robot.model.totalstatistics.TotalStatisticsBuilder method*), 289
end_for_iteration() (*robot.model.visitor.SuiteVisitor method*), 295
end_for_iteration() (*robot.output.console.dotted.StatusReporter method*), 299
end_for_iteration() (*robot.output.xmllogger.XmlLogger method*), 314
end_for_iteration() (*robot.reporting.outputwriter.OutputWriter method*), 396
end_for_iteration() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 403
end_for_iteration() (*robot.result.configurer.SuiteConfigurer method*), 409
end_for_iteration() (*robot.result.keywordremover.AllKeywordsRemover method*), 416
end_for_iteration() (*robot.result.keywordremover.ByNameKeywordRemover method*), 424
end_for_iteration() (*robot.result.keywordremover.ByTagKeywordRemover method*), 428
end_for_iteration() (*robot.result.keywordremover.ForLoopItemsRemover method*), 432
end_for_iteration() (*robot.result.keywordremover.PassedKeywordRemover method*), 420
end_for_iteration() (*robot.result.keywordremover.WaitUntilKeywordSucceedsRemover method*), 441
end_for_iteration() (*robot.result.keywordremover.WarningAndErrorFinder method*), 445
end_for_iteration() (*robot.result.keywordremover.WhileLoopItemsRemover method*), 437
end_for_iteration() (*robot.result.merger.Merger method*), 450
end_for_iteration() (*robot.result.messagefilter.MessageFilter method*), 454
end_for_iteration() (*robot.result.resultbuilder.RemoveKeywords method*), 501
end_for_iteration() (*robot.result.suiteteardownfailed.SuiteTeardownFailed method*), 510
end_for_iteration() (*robot.result.suiteteardownfailed.SuiteTeardownFailureHandler method*), 506
end_for_iteration()

(*robot.result.visitor.ResultVisitor method*), 515
end_for_iteration() (*robot.running.randomizer.Randomizer method*), 579
end_for_iteration() (*robot.running.suiterunner.SuiteRunner method*), 585
end_if() (*robot.conf.gatherfailed.GatherFailedSuites method*), 28
end_if() (*robot.conf.gatherfailed.GatherFailedTests method*), 24
end_if() (*robot.model.configurer.SuiteConfigurer method*), 235
end_if() (*robot.model.filter.EmptySuiteRemover method*), 251
end_if() (*robot.model.filter.Filter method*), 255
end_if() (*robot.model.modifier.ModelModifier method*), 266
end_if() (*robot.model.statistics.StatisticsBuilder method*), 271
end_if() (*robot.model.tagsetter.TagSetter method*), 278
end_if() (*robot.model.totalstatistics.TotalStatisticsBuilder method*), 289
end_if() (*robot.model.visitor.SuiteVisitor method*), 295
end_if() (*robot.output.console.dotted.StatusReporter method*), 299
end_if() (*robot.output.xmllogger.XmlLogger method*), 313
end_if() (*robot.reporting.outputwriter.OutputWriter method*), 396
end_if() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 403
end_if() (*robot.result.configurer.SuiteConfigurer method*), 409
end_if() (*robot.result.keywordremover.AllKeywordsRemover method*), 416
end_if() (*robot.result.keywordremover.ByNameKeywordRemover method*), 424
end_if() (*robot.result.keywordremover.ByTagKeywordRemover method*), 428
end_if() (*robot.result.keywordremover.ForLoopItemsRemover method*), 432
end_if() (*robot.result.keywordremover.PassedKeywordRemover method*), 420
end_if() (*robot.result.keywordremover.WaitUntilKeywordSucceedsRemover method*), 441
end_if() (*robot.result.keywordremover.WarningAndErrorFinder method*), 445
end_if() (*robot.result.keywordremover.WhileLoopItemsRemover method*), 437
end_if() (*robot.result.merger.Merger method*), 450
end_if() (*robot.result.messagefilter.MessageFilter method*), 454
end_if() (*robot.result.resultbuilder.RemoveKeywords method*), 501
end_if() (*robot.result.suiteteardownfailed.SuiteTeardownFailed method*), 510
end_if() (*robot.result.suiteteardownfailed.SuiteTeardownFailureHandle method*), 506
end_if() (*robot.result.visitor.ResultVisitor method*), 515
end_if() (*robot.running.randomizer.Randomizer method*), 579
end_if() (*robot.running.suiterunner.SuiteRunner method*), 585
end_if_branch() (*robot.conf.gatherfailed.GatherFailedSuites method*), 28
end_if_branch() (*robot.conf.gatherfailed.GatherFailedTests method*), 24
end_if_branch() (*robot.model.configurer.SuiteConfigurer method*), 235
end_if_branch() (*robot.model.filter.EmptySuiteRemover method*), 251
end_if_branch() (*robot.model.filter.Filter method*), 255
end_if_branch() (*robot.model.modifier.ModelModifier method*), 266
end_if_branch() (*robot.model.statistics.StatisticsBuilder method*), 271
end_if_branch() (*robot.model.tagsetter.TagSetter method*), 278
end_if_branch() (*robot.model.totalstatistics.TotalStatisticsBuilder method*), 289
end_if_branch() (*robot.model.visitor.SuiteVisitor method*), 296
end_if_branch() (*robot.output.console.dotted.StatusReporter method*), 299
end_if_branch() (*robot.output.xmllogger.XmlLogger method*), 314
end_if_branch() (*robot.reporting.outputwriter.OutputWriter method*), 396
end_if_branch() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 403
end_if_branch() (*robot.result.configurer.SuiteConfigurer method*), 409
end_if_branch() (*robot.result.keywordremover.AllKeywordsRemover method*), 416
end_if_branch() (*robot.result.keywordremover.ByNameKeywordRemo*
Remover method), 424
end_if_branch() (*robot.result.keywordremover.ByTagKeywordRemove method*), 428
end_if_branch() (*robot.result.keywordremover.ForLoopItemsRemover method*), 433
end_if_branch() (*robot.result.keywordremover.PassedKeywordRemov method*), 420

end_if_branch() (robot.result.keywordremover.WaitUntilKeywordSucceedsRemover method), 441
end_if_branch() (robot.result.keywordremover.WarningAndErrorFinder method), 445
end_if_branch() (robot.result.keywordremover.WhileLoopItemsRemover method), 437
end_if_branch() (robot.result.merger.Merger method), 450
end_if_branch() (robot.result.messagefilter.MessageFilter method), 454
end_if_branch() (robot.result.resultbuilder.RemoveKeywords method), 501
end_if_branch() (robot.result.suiteteardownfailed.SuiteTeardownFailed method), 510
end_if_branch() (robot.result.suiteteardownfailed.SuiteTeardownFailureHandler method), 506
end_if_branch() (robot.result.visitor.ResultVisitor method), 515
end_if_branch() (robot.running.randomizer.Randomizer method), 579
end_if_branch() (robot.running.suiterunner.SuiteRunner method), 585
end_keyword() (robot.conf.gatherfailed.GatherFailedSuites method), 28
end_keyword() (robot.conf.gatherfailed.GatherFailedTests method), 24
end_keyword() (robot.model.configurer.SuiteConfigurer method), 235
end_keyword() (robot.model.filter.EmptySuiteRemover method), 251
end_keyword() (robot.model.filter.Filter method), 255
end_keyword() (robot.model.modifier.ModelModifier method), 266
end_keyword() (robot.model.statistics.StatisticsBuilder method), 271
end_keyword() (robot.model.tagsetter.TagSetter method), 278
end_keyword() (robot.model.totalstatistics.TotalStatisticsBuilder method), 289
end_keyword() (robot.model.visitor.SuiteVisitor method), 295
end_keyword() (robot.output.console.dotted.StatusReporter method), 299
end_keyword() (robot.output.console.verbose.VerboseOutput method), 304
end_keyword() (robot.output.filelogger.FileLogger method), 305
end_keyword() (robot.output.listeners.Listeners method), 307
end_keyword() (robot.output.logger.Logger method), 308
end_keyword() (robot.output.logger.LoggerProxy method), 309
end_keyword() (robot.output.output.Output method), 311
end_keyword() (robot.output.xmllogger.XmlLogger method), 313
end_keyword() (robot.reporting.outputwriter.OutputWriter method), 396
end_keyword() (robot.reporting.xunitwriter.XUnitFileWriter method), 403
end_keyword() (robot.result.configurer.SuiteConfigurer method), 409
end_keyword() (robot.result.keywordremover.AllKeywordsRemover method), 416
end_keyword() (robot.result.keywordremover.ByNameKeywordRemover method), 424
end_keyword() (robot.result.keywordremover.ByTagKeywordRemover method), 428
end_keyword() (robot.result.keywordremover.ForLoopItemsRemover method), 433
end_keyword() (robot.result.keywordremover.PassedKeywordRemover method), 420
end_keyword() (robot.result.keywordremover.WaitUntilKeywordSucceeds method), 441
end_keyword() (robot.result.keywordremover.WarningAndErrorFinder method), 446
end_keyword() (robot.result.keywordremover.WhileLoopItemsRemover method), 437
end_keyword() (robot.result.merger.Merger method), 450
end_keyword() (robot.result.messagefilter.MessageFilter method), 454
end_keyword() (robot.result.resultbuilder.RemoveKeywords method), 501
end_keyword() (robot.result.suiteteardownfailed.SuiteTeardownFailed method), 510
end_keyword() (robot.result.suiteteardownfailed.SuiteTeardownFailureHandler method), 506
end_keyword() (robot.result.visitor.ResultVisitor method), 515
end_keyword() (robot.running.randomizer.Randomizer method), 579
end_keyword() (robot.running.suiterunner.SuiteRunner method), 585
end_keyword() (robot.variables.scopes.SetVariables method), 610
end_keyword() (robot.variables.scopes.VariableScopes method), 609
end_lineno (robot.parsing.model.blocks.Block attribute), 340
end_lineno (robot.parsing.model.blocks.CommentSection attribute), 342
end_lineno (robot.parsing.model.blocks.File attribute), 341
end_lineno (robot.parsing.model.blocks.For attribute), 344

end_lineno (*robot.parsing.model.blocks.HeaderAndBody attribute*), 341
end_lineno (*robot.parsing.model.blocks.If attribute*), 343
end_lineno (*robot.parsing.model.blocks.Keyword attribute*), 343
end_lineno (*robot.parsing.model.blocks.KeywordSection attribute*), 342
end_lineno (*robot.parsing.model.blocks.Section attribute*), 341
end_lineno (*robot.parsing.model.blocks.SettingSection attribute*), 341
end_lineno (*robot.parsing.model.blocks.TestCase attribute*), 343
end_lineno (*robot.parsing.model.blocks.TestCaseSection attribute*), 342
end_lineno (*robot.parsing.model.blocks.Try attribute*), 344
end_lineno (*robot.parsing.model.blocks.VariableSection attribute*), 342
end_lineno (*robot.parsing.model.blocks.While attribute*), 344
end_lineno (*robot.parsing.model.statements.Arguments attribute*), 369
end_lineno (*robot.parsing.model.statements.Break attribute*), 384
end_lineno (*robot.parsing.model.statements.Comment attribute*), 385
end_lineno (*robot.parsing.model.statements.Config attribute*), 386
end_lineno (*robot.parsing.model.statements.Continue attribute*), 383
end_lineno (*robot.parsing.model.statements.DefaultTags attribute*), 356
end_lineno (*robot.parsing.model.statements.Documentation attribute*), 353
end_lineno (*robot.parsing.model.statements.DocumentationOrMetadata attribute*), 346
end_lineno (*robot.parsing.model.statements.ElseHeader attribute*), 376
end_lineno (*robot.parsing.model.statements.ElseIfHeader attribute*), 375
end_lineno (*robot.parsing.model.statements.EmptyLine attribute*), 388
end_lineno (*robot.parsing.model.statements.End attribute*), 380
end_lineno (*robot.parsing.model.statements.Error attribute*), 387
end_lineno (*robot.parsing.model.statements.ExceptHeader attribute*), 379
end_lineno (*robot.parsing.model.statements.FinallyHeader attribute*), 379
end_lineno (*robot.parsing.model.statements.Fixture attribute*), 349
end_lineno (*robot.parsing.model.statements.ForceTags attribute*), 355
end_lineno (*robot.parsing.model.statements.ForHeader attribute*), 372
end_lineno (*robot.parsing.model.statements.IfElseHeader attribute*), 373
end_lineno (*robot.parsing.model.statements.IfHeader attribute*), 374
end_lineno (*robot.parsing.model.statements.InlineIfHeader attribute*), 374
end_lineno (*robot.parsing.model.statements.KeywordCall attribute*), 370
end_lineno (*robot.parsing.model.statements.KeywordName attribute*), 364
end_lineno (*robot.parsing.model.statements.KeywordTags attribute*), 356
end_lineno (*robot.parsing.model.statements.LibraryImport attribute*), 351
end_lineno (*robot.parsing.model.statements.LoopControl attribute*), 383
end_lineno (*robot.parsing.model.statements.Metadata attribute*), 354
end_lineno (*robot.parsing.model.statements.MultiValue attribute*), 348
end_lineno (*robot.parsing.model.statements.NoArgumentHeader attribute*), 377
end_lineno (*robot.parsing.model.statements.ResourceImport attribute*), 352
end_lineno (*robot.parsing.model.statements.Return attribute*), 369
end_lineno (*robot.parsing.model.statements.ReturnStatement attribute*), 382
end_lineno (*robot.parsing.model.statements.SectionHeader attribute*), 350
end_lineno (*robot.parsing.model.statements.Setup attribute*), 365
end_lineno (*robot.parsing.model.statements.SingleValue attribute*), 347
end_lineno (*robot.parsing.model.statements.Statement attribute*), 346
end_lineno (*robot.parsing.model.statements.SuiteSetup attribute*), 357
end_lineno (*robot.parsing.model.statements.SuiteTeardown attribute*), 358
end_lineno (*robot.parsing.model.statements.Tags attribute*), 366
end_lineno (*robot.parsing.model.statements.Teardown attribute*), 365
end_lineno (*robot.parsing.model.statements.Template attribute*), 367
end_lineno (*robot.parsing.model.statements.TemplateArguments attribute*), 371
end_lineno (*robot.parsing.model.statements.TestCaseName attribute*), 363

end_lineno (*robot.parsing.model.statements.TestSetup attribute*), 359
end_lineno (*robot.parsing.model.statements.TestTeardown attribute*), 360
end_lineno (*robot.parsing.model.statements.TestTemplate attribute*), 361
end_lineno (*robot.parsing.model.statements.TestTimeout attribute*), 361
end_lineno (*robot.parsing.model.statements.Timeout attribute*), 368
end_lineno (*robot.parsing.model.statements.TryHeader attribute*), 378
end_lineno (*robot.parsing.model.statements.Variable attribute*), 362
end_lineno (*robot.parsing.model.statements.VariablesImport attribute*), 352
end_lineno (*robot.parsing.model.statements.WhileHeader attribute*), 381
end_loggers (*robot.output.logger.Logger attribute*), 308
end_message() (*robot.conf.gatherfailed.GatherFailedSuites method*), 28
end_message() (*robot.conf.gatherfailed.GatherFailedTests method*), 24
end_message() (*robot.model.configurer.SuiteConfigurer method*), 235
end_message() (*robot.model.filter.EmptySuiteRemover method*), 251
end_message() (*robot.model.filter.Filter method*), 255
end_message() (*robot.model.modifier.ModelModifier method*), 266
end_message() (*robot.model.statistics.StatisticsBuilder method*), 271
end_message() (*robot.model.tagsetter.TagSetter method*), 278
end_message() (*robot.model.totalstatistics.TotalStatisticsBuilder method*), 289
end_message() (*robot.model.visitor.SuiteVisitor method*), 298
end_message() (*robot.output.console.dotted.StatusReporter method*), 299
end_message() (*robot.output.xmllogger.XmlLogger method*), 316
end_message() (*robot.reporting.outputwriter.OutputWriter method*), 396
end_message() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 403
end_message() (*robot.result.configurer.SuiteConfigurer method*), 409
end_message() (*robot.result.keywordremover.AllKeywordsRemover method*), 416
end_message() (*robot.result.keywordremover.ByNameKeywordRemover method*), 424
end_message() (*robot.result.keywordremover.ByTagKeywordRemover method*), 429
end_message() (*robot.result.keywordremover.ForLoopItemsRemover method*), 433
end_message() (*robot.result.keywordremover.PassedKeywordRemover method*), 420
end_message() (*robot.result.keywordremover.WaitUntilKeywordSucceed method*), 441
end_message() (*robot.result.keywordremover.WarningAndErrorFinder method*), 446
end_message() (*robot.result.keywordremover.WhileLoopItemsRemover method*), 437
end_message() (*robot.result.merger.Merger method*), 450
end_message() (*robot.result.messagefilter.MessageFilter method*), 454
end_message() (*robot.result.resultbuilder.RemoveKeywords method*), 501
end_message() (*robot.result.suiteteardownfailed.SuiteTeardownFailed method*), 510
end_message() (*robot.result.suiteteardownfailed.SuiteTeardownFailure method*), 506
end_message() (*robot.result.visitor.ResultVisitor method*), 515
end_message() (*robot.running.randomizer.Randomizer method*), 579
end_message() (*robot.running.suiterunner.SuiteRunner method*), 585
end_result() (*robot.output.xmllogger.XmlLogger method*), 316
end_result() (*robot.reporting.outputwriter.OutputWriter method*), 396
end_result() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 403
end_result() (*robot.result.visitor.ResultVisitor method*), 514
end_return() (*robot.conf.gatherfailed.GatherFailedSuites method*), 28
end_return() (*robot.conf.gatherfailed.GatherFailedTests method*), 24
end_return() (*robot.model.configurer.SuiteConfigurer method*), 235
end_return() (*robot.model.filter.EmptySuiteRemover method*), 251
end_return() (*robot.model.filter.Filter method*), 255
end_return() (*robot.model.modifier.ModelModifier method*), 266
end_return() (*robot.model.statistics.StatisticsBuilder method*), 271
end_return() (*robot.model.tagsetter.TagSetter method*), 278
end_return() (*robot.model.totalstatistics.TotalStatisticsBuilder method*), 289
end_return() (*robot.model.visitor.SuiteVisitor*

method), 297
end_return() (*robot.output.console.dotted.StatusReporter method*), 300
end_return() (*robot.output.xmllogger.XmlLogger method*), 315
end_return() (*robot.reporting.outputwriter.OutputWriter method*), 396
end_return() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 403
end_return() (*robot.result.configurer.SuiteConfigurer method*), 409
end_return() (*robot.result.keywordremover.AllKeywordsRemover method*), 416
end_return() (*robot.result.keywordremover.ByNameKeywordRemover method*), 424
end_return() (*robot.result.keywordremover.ByTagKeywordRemover method*), 429
end_return() (*robot.result.keywordremover.ForLoopItemsRemover method*), 433
end_return() (*robot.result.keywordremover.PassedKeywordRemover method*), 420
end_return() (*robot.result.keywordremover.WaitUntilKeywordSucceedsRemover method*), 441
end_return() (*robot.result.keywordremover.WarningAndErrorFinder method*), 446
end_return() (*robot.result.keywordremover.WhileLoopItemsRemover method*), 437
end_return() (*robot.result.merger.Merger method*), 450
end_return() (*robot.result.messagefilter.MessageFilter method*), 454
end_return() (*robot.result.resultbuilder.RemoveKeywords method*), 502
end_return() (*robot.result.suiteteardownfailed.SuiteTeardownFailed method*), 510
end_return() (*robot.result.suiteteardownfailed.SuiteTeardownFailureHandler method*), 506
end_return() (*robot.result.visitor.ResultVisitor method*), 515
end_return() (*robot.running.randomizer.Randomizer method*), 579
end_return() (*robot.running.suiterunner.SuiteRunner method*), 586
end_splitting() (*robot.reporting.jsbuildingcontext.JsBuildingContext method*), 394
end_stat() (*robot.output.xmllogger.XmlLogger method*), 316
end_stat() (*robot.reporting.outputwriter.OutputWriter method*), 396
end_stat() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 403
end_stat() (*robot.result.visitor.ResultVisitor method*), 514
end_statistics() (*robot.output.xmllogger.XmlLogger method*), 316
end_statistics() (*robot.reporting.outputwriter.OutputWriter method*), 396
end_statistics() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 403
end_statistics() (*robot.result.visitor.ResultVisitor method*), 514
end_suite() (*robot.conf.gatherfailed.GatherFailedSuites method*), 28
end_suite() (*robot.conf.gatherfailed.GatherFailedTests method*), 24
end_suite() (*robot.model.configurer.SuiteConfigurer method*), 235
end_suite() (*robot.model.filter.EmptySuiteRemover method*), 250
end_suite() (*robot.model.filter.Filter method*), 255
end_suite() (*robot.model.modifier.ModelModifier method*), 266
end_suite() (*robot.model.statistics.StatisticsBuilder method*), 270
end_suite() (*robot.model.suitestatistics.SuiteStatisticsBuilder method*), 277
end_suite() (*robot.model.tagsetter.TagSetter method*), 278
end_suite() (*robot.model.totalstatistics.TotalStatisticsBuilder method*), 289
end_suite() (*robot.model.visitor.SuiteVisitor method*), 294
end_suite() (*robot.output.console.dotted.DottedOutput method*), 299
end_suite() (*robot.output.console.dotted.StatusReporter method*), 300
end_suite() (*robot.output.console.verbose.VerboseOutput method*), 304
end_suite() (*robot.output.filelogger.FileLogger method*), 305
end_suite() (*robot.output.logger.Logger method*), 308
end_suite() (*robot.output.output.Output method*), 311
end_suite() (*robot.output.xmllogger.XmlLogger method*), 316
end_suite() (*robot.reporting.outputwriter.OutputWriter method*), 397
end_suite() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 402
end_suite() (*robot.result.configurer.SuiteConfigurer method*), 409
end_suite() (*robot.result.keywordremover.AllKeywordsRemover method*), 416
end_suite() (*robot.result.keywordremover.ByNameKeywordRemover method*), 425
end_suite() (*robot.result.keywordremover.ByTagKeywordRemover method*), 429

end_suite() (*robot.result.keywordremover.ForLoopItemsRemover method*), 433
end_suite() (*robot.result.keywordremover.PassedKeywordRemover method*), 421
end_suite() (*robot.result.keywordremover.WaitUntilKeywordSucceedsRemover method*), 441
end_suite() (*robot.result.keywordremover.WarningAndErrorFinder method*), 446
end_suite() (*robot.result.keywordremover.WhileLoopItemsRemover method*), 437
end_suite() (*robot.result.merger.Merger method*), 449
end_suite() (*robot.result.messagefilter.MessageFilter method*), 454
end_suite() (*robot.result.resultbuilder.RemoveKeywords method*), 502
end_suite() (*robot.result.suiteteardownfailed.SuiteTeardownFailed method*), 510
end_suite() (*robot.result.suiteteardownfailed.SuiteTeardownFailureHandler method*), 505
end_suite() (*robot.result.visitor.ResultVisitor method*), 515
end_suite() (*robot.running.context.ExecutionContexts method*), 550
end_suite() (*robot.running.libraryscopes.GlobalScope method*), 553
end_suite() (*robot.running.libraryscopes.TestCaseScope method*), 553
end_suite() (*robot.running.libraryscopes.TestSuiteScope method*), 553
end_suite() (*robot.running.namespace.Namespace method*), 577
end_suite() (*robot.running.randomizer.Randomizer method*), 579
end_suite() (*robot.running.suiterunner.SuiteRunner method*), 585
end_suite() (*robot.variables.scopes.SetVariables method*), 610
end_suite() (*robot.variables.scopes.VariableScopes method*), 609
end_suite_statistics() (*robot.output.xmllogger.XmlLogger method*), 316
end_suite_statistics() (*robot.reporting.outputwriter.OutputWriter method*), 397
end_suite_statistics() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 403
end_suite_statistics() (*robot.result.visitor.ResultVisitor method*), 514
end_tag_statistics() (*robot.output.xmllogger.XmlLogger method*), 316
end_tag_statistics() (*robot.reporting.outputwriter.OutputWriter method*), 397
end_tag_statistics() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 403
end_tag_statistics() (*robot.result.visitor.ResultVisitor method*), 514
end_test() (*robot.conf.gatherfailed.GatherFailedSuites method*), 28
end_test() (*robot.conf.gatherfailed.GatherFailedTests method*), 24
end_test() (*robot.model.configurer.SuiteConfigurer method*), 235
end_test() (*robot.model.filter.EmptySuiteRemover method*), 251
end_test() (*robot.model.filter.Filter method*), 255
end_test() (*robot.model.modifier.ModelModifier method*), 266
end_test() (*robot.model.statistics.StatisticsBuilder method*), 271
end_test() (*robot.model.tagsetter.TagSetter method*), 278
end_test() (*robot.model.totalstatistics.TotalStatisticsBuilder method*), 289
end_test() (*robot.model.visitor.SuiteVisitor method*), 294
end_test() (*robot.output.console.dotted.DottedOutput method*), 299
end_test() (*robot.output.console.dotted.StatusReporter method*), 300
end_test() (*robot.output.console.verbose.VerboseOutput method*), 304
end_test() (*robot.output.filelogger.FileLogger method*), 305
end_test() (*robot.output.logger.Logger method*), 308
end_test() (*robot.output.output.Output method*), 311
end_test() (*robot.output.xmllogger.XmlLogger method*), 315
end_test() (*robot.reporting.outputwriter.OutputWriter method*), 397
end_test() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 403
end_test() (*robot.result.configurer.SuiteConfigurer method*), 409
end_test() (*robot.result.keywordremover.AllKeywordsRemover method*), 416
end_test() (*robot.result.keywordremover.ByNameKeywordRemover method*), 425
end_test() (*robot.result.keywordremover.ByTagKeywordRemover method*), 429
end_test() (*robot.result.keywordremover.ForLoopItemsRemover*

method), 433
end_test() (*robot.result.keywordremover.PassedKeywordRemover method*), 421
end_test() (*robot.result.keywordremover.WaitUntilKeywordSucceedsRemover method*), 441
end_test() (*robot.result.keywordremover.WarningAndErrorFinder method*), 446
end_test() (*robot.result.keywordremover.WhileLoopItemsRemover method*), 437
end_test() (*robot.result.merger.Merger method*), 450
end_test() (*robot.result.messagefilter.MessageFilter method*), 454
end_test() (*robot.result.resultbuilder.RemoveKeywords method*), 502
end_test() (*robot.result.suiteteardownfailed.SuiteTeardownFailed method*), 510
end_test() (*robot.result.suiteteardownfailed.SuiteTeardownFailureHandler method*), 506
end_test() (*robot.result.visitor.ResultVisitor method*), 515
end_test() (*robot.running.libraryscopes.GlobalScope method*), 553
end_test() (*robot.running.libraryscopes.TestCaseScope method*), 553
end_test() (*robot.running.libraryscopes.TestSuiteScope method*), 553
end_test() (*robot.running.namespace.Namespace method*), 577
end_test() (*robot.running.randomizer.Randomizer method*), 579
end_test() (*robot.running.suiterunner.SuiteRunner method*), 586
end_test() (*robot.variables.scopes.SetVariables method*), 610
end_test() (*robot.variables.scopes.VariableScopes method*), 609
end_total_statistics() (*robot.output.xmllogger.XmlLogger method*), 316
end_total_statistics() (*robot.reporting.outputwriter.OutputWriter method*), 397
end_total_statistics() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 404
end_total_statistics() (*robot.result.visitor.ResultVisitor method*), 514
end_try() (*robot.conf.gatherfailed.GatherFailedSuites method*), 28
end_try() (*robot.conf.gatherfailed.GatherFailedTests method*), 24
end_try() (*robot.model.configurer.SuiteConfigurer method*), 235
end_try() (*robot.model.filter.EmptySuiteRemover method*), 251
end_try() (*robot.model.filter.Filter method*), 255
end_try() (*robot.model.modifier.ModelModifier method*), 266
end_try() (*robot.model.statistics.StatisticsBuilder method*), 271
end_try() (*robot.model.tagsetter.TagSetter method*), 279
end_try() (*robot.model.totalstatistics.TotalStatisticsBuilder method*), 290
end_try() (*robot.model.visitor.SuiteVisitor method*), 296
end_try() (*robot.output.console.dotted.StatusReporter method*), 300
end_try() (*robot.output.xmllogger.XmlLogger method*), 314
end_try() (*robot.reporting.outputwriter.OutputWriter method*), 397
end_try() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 404
end_try() (*robot.result.configurer.SuiteConfigurer method*), 409
end_try() (*robot.result.keywordremover.AllKeywordsRemover method*), 416
end_try() (*robot.result.keywordremover.ByNameKeywordRemover method*), 425
end_try() (*robot.result.keywordremover.ByTagKeywordRemover method*), 429
end_try() (*robot.result.keywordremover.ForLoopItemsRemover method*), 433
end_try() (*robot.result.keywordremover.PassedKeywordRemover method*), 421
end_try() (*robot.result.keywordremover.WaitUntilKeywordSucceedsRem method*), 441
end_try() (*robot.result.keywordremover.WarningAndErrorFinder method*), 446
end_try() (*robot.result.keywordremover.WhileLoopItemsRemover method*), 437
end_try() (*robot.result.merger.Merger method*), 450
end_try() (*robot.result.messagefilter.MessageFilter method*), 454
end_try() (*robot.result.resultbuilder.RemoveKeywords method*), 502
end_try() (*robot.result.suiteteardownfailed.SuiteTeardownFailed method*), 510
end_try() (*robot.result.suiteteardownfailed.SuiteTeardownFailureHandl method*), 506
end_try() (*robot.result.visitor.ResultVisitor method*), 515
end_try() (*robot.running.randomizer.Randomizer method*), 579
end_try() (*robot.running.suiterunner.SuiteRunner method*), 586

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), 235
end_try_branch() (robot.model.filter.EmptySuiteRemover method), 251
end_try_branch() (robot.model.filter.Filter method), 255
end_try_branch() (robot.model.modifier.ModelModifier method), 266
end_try_branch() (robot.model.statistics.StatisticsBuilder method), 271
end_try_branch() (robot.model.tagsetter.TagSetter method), 279
end_try_branch() (robot.model.totalstatistics.TotalStatisticsBuilder method), 290
end_try_branch() (robot.model.visitor.SuiteVisitor method), 296
end_try_branch() (robot.output.console.dotted.StatusReporter method), 300
end_try_branch() (robot.output.xmllogger.XmlLogger method), 314
end_try_branch() (robot.reporting.outputwriter.OutputWriter method), 397
end_try_branch() (robot.reporting.xunitwriter.XUnitFileWriter method), 404
end_try_branch() (robot.result.configurer.SuiteConfigurer method), 409
end_try_branch() (robot.result.keywordremover.AllKeywordsRemover method), 417
end_try_branch() (robot.result.keywordremover.ByNameKeywordRemover method), 425
end_try_branch() (robot.result.keywordremover.ByTagKeywordRemover method), 429
end_try_branch() (robot.result.keywordremover.ForLoopItemsRemover method), 433
end_try_branch() (robot.result.keywordremover.PassedKeywordRemover method), 421
end_try_branch() (robot.result.keywordremover.WaitUntilKeywordSucceedsRemover method), 441
end_try_branch() (robot.result.keywordremover.WarningAndErrorFinder method), 446
end_try_branch() (robot.result.keywordremover.WhileLoopItemsRemover method), 437
end_try_branch() (robot.result.merger.Merger method), 450
end_try_branch() (robot.result.messagefilter.MessageFilter method), 454
end_try_branch() (robot.result.resultbuilder.RemoveKeywords method), 502
end_try_branch() (robot.result.suiteteardownfailed.SuiteTeardownFailed method), 510
end_try_branch() (robot.result.suiteteardownfailed.SuiteTeardownFailureHandler method), 506
end_try_branch() (robot.result.visitor.ResultVisitor method), 515
end_try_branch() (robot.running.randomizer.Randomizer method), 579
end_try_branch() (robot.running.suiterunner.SuiteRunner method), 586
end_user_keyword() (robot.running.namespace.Namespace method), 578
end_while() (robot.conf.gatherfailed.GatherFailedSuites method), 29
end_while() (robot.conf.gatherfailed.GatherFailedTests method), 24
end_while() (robot.model.configurer.SuiteConfigurer method), 235
end_while() (robot.model.filter.EmptySuiteRemover method), 251
end_while() (robot.model.filter.Filter method), 255
end_while() (robot.model.modifier.ModelModifier method), 266
end_while() (robot.model.statistics.StatisticsBuilder method), 271
end_while() (robot.model.tagsetter.TagSetter method), 279
end_while() (robot.model.totalstatistics.TotalStatisticsBuilder method), 290
end_while() (robot.model.visitor.SuiteVisitor method), 297
end_while() (robot.output.console.dotted.StatusReporter method), 300
end_while() (robot.output.xmllogger.XmlLogger method), 315
end_while() (robot.reporting.outputwriter.OutputWriter method), 397
end_while() (robot.reporting.xunitwriter.XUnitFileWriter method), 404
end_while() (robot.result.configurer.SuiteConfigurer method), 409
end_while() (robot.result.keywordremover.AllKeywordsRemover method), 417
end_while() (robot.result.keywordremover.ByNameKeywordRemover method), 425
end_while() (robot.result.keywordremover.ByTagKeywordRemover method), 429
end_while() (robot.result.keywordremover.ForLoopItemsRemover method), 433
end_while() (robot.result.keywordremover.PassedKeywordRemover method), 421
end_while() (robot.result.keywordremover.WaitUntilKeywordSucceedsR method), 442
end_while() (robot.result.keywordremover.WarningAndErrorFinder method), 446

end_while() (*robot.result.keywordremover.WhileLoopItemsRemover method*), 437
end_while() (*robot.result.merger.Merger method*), 450
end_while() (*robot.result.messagefilter.MessageFilter method*), 454
end_while() (*robot.result.resultbuilder.RemoveKeywords method*), 502
end_while() (*robot.result.suiteteardownfailed.SuiteTeardownFailed method*), 510
end_while() (*robot.result.suiteteardownfailed.SuiteTeardownFailureHandler method*), 506
end_while() (*robot.result.visitor.ResultVisitor method*), 515
end_while() (*robot.running.randomizer.Randomizer method*), 580
end_while() (*robot.running.suiterunner.SuiteRunner method*), 586
end_while_iteration() (*robot.conf.gatherfailed.GatherFailedSuites method*), 29
end_while_iteration() (*robot.conf.gatherfailed.GatherFailedTests method*), 24
end_while_iteration() (*robot.model.configurer.SuiteConfigurer method*), 235
end_while_iteration() (*robot.model.filter.EmptySuiteRemover method*), 251
end_while_iteration() (*robot.model.filter.Filter method*), 255
end_while_iteration() (*robot.model.modifier.ModelModifier method*), 267
end_while_iteration() (*robot.model.statistics.StatisticsBuilder method*), 272
end_while_iteration() (*robot.model.tagsetter.TagSetter method*), 279
end_while_iteration() (*robot.model.totalstatistics.TotalStatisticsBuilder method*), 290
end_while_iteration() (*robot.model.visitor.SuiteVisitor method*), 297
end_while_iteration() (*robot.output.console.dotted.StatusReporter method*), 300
end_while_iteration() (*robot.output.xmllogger.XmlLogger method*), 315
end_while_iteration() (*robot.reporting.outputwriter.OutputWriter method*), 397
end_while_iteration() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 404
end_while_iteration() (*robot.result.configurer.SuiteConfigurer method*), 409
end_while_iteration() (*robot.result.keywordremover.AllKeywordsRemover method*), 417
end_while_iteration() (*robot.result.keywordremover.ByNameKeywordRemover method*), 425
end_while_iteration() (*robot.result.keywordremover.ByTagKeywordRemover method*), 429
end_while_iteration() (*robot.result.keywordremover.ForLoopItemsRemover method*), 433
end_while_iteration() (*robot.result.keywordremover.PassedKeywordRemover method*), 421
end_while_iteration() (*robot.result.keywordremover.WaitUntilKeywordSucceedsRemover method*), 442
end_while_iteration() (*robot.result.keywordremover.WarningAndErrorFinder method*), 446
end_while_iteration() (*robot.result.keywordremover.WhileLoopItemsRemover method*), 437
end_while_iteration() (*robot.result.merger.Merger method*), 450
end_while_iteration() (*robot.result.messagefilter.MessageFilter method*), 454
end_while_iteration() (*robot.result.resultbuilder.RemoveKeywords method*), 502
end_while_iteration() (*robot.result.suiteteardownfailed.SuiteTeardownFailed method*), 510
end_while_iteration() (*robot.result.suiteteardownfailed.SuiteTeardownFailureHandler method*), 506
end_while_iteration() (*robot.result.visitor.ResultVisitor method*), 515
end_while_iteration() (*robot.running.randomizer.Randomizer method*), 580
end_while_iteration() (*robot.running.suiterunner.SuiteRunner*

method), 586
EndKeywordArguments (*class in robot.output.listenerarguments*), 306
EndLexer (*class in robot.parsing.lexer.statementlexers*), 332
EndSuiteArguments (*class in robot.output.listenerarguments*), 306
EndTestArguments (*class in robot.output.listenerarguments*), 306
endtime (*robot.result.model.Break attribute*), 488
endtime (*robot.result.model.Continue attribute*), 485
endtime (*robot.result.model.For attribute*), 467
endtime (*robot.result.model.ForIteration attribute*), 465
endtime (*robot.result.model.If attribute*), 476
endtime (*robot.result.model.IfBranch attribute*), 474
endtime (*robot.result.model.Keyword attribute*), 490
endtime (*robot.result.model.Return attribute*), 483
endtime (*robot.result.model.TestCase attribute*), 493
endtime (*robot.result.model.TestSuite attribute*), 496
endtime (*robot.result.model.Try attribute*), 481
endtime (*robot.result.model.TryBranch attribute*), 478
endtime (*robot.result.model.While attribute*), 472
endtime (*robot.result.model.WhileIteration attribute*), 469
ENUM (*robot.libdocpkg.datatypes.TypeDoc attribute*), 70
EnumConverter (*class in robot.running.arguments.typeconverters*), 532
EnumMember (*class in robot.libdocpkg.datatypes*), 70
environment_variable_should_be_set() (*robot.libraries.OperatingSystem.OperatingSystem method*), 119
environment_variable_should_not_be_set() (*robot.libraries.OperatingSystem.OperatingSystem method*), 119
EnvironmentFinder (*class in robot.variables.finders*), 608
EOL (*robot.parsing.lexer.tokens.END attribute*), 338
EOL (*robot.parsing.lexer.tokens.EOS attribute*), 336
EOL (*robot.parsing.lexer.tokens.Token attribute*), 335
EOS (*class in robot.parsing.lexer.tokens*), 336
EOS (*robot.parsing.lexer.tokens.END attribute*), 338
EOS (*robot.parsing.lexer.tokens.EOS attribute*), 336
EOS (*robot.parsing.lexer.tokens.Token attribute*), 335
eq() (*in module robot.utils.match*), 603
Error, 13
Error (*class in robot.parsing.model.statements*), 387
ERROR (*robot.parsing.lexer.tokens.END attribute*), 338
error (*robot.parsing.lexer.tokens.END attribute*), 340
ERROR (*robot.parsing.lexer.tokens.EOS attribute*), 336
error (*robot.parsing.lexer.tokens.EOS attribute*), 338
ERROR (*robot.parsing.lexer.tokens.Token attribute*), 335
error (*robot.parsing.lexer.tokens.Token attribute*), 335
error (*robot.running.model.Break attribute*), 569
error (*robot.running.model.Continue attribute*), 567
error (*robot.running.model.For attribute*), 557
error (*robot.running.model.If attribute*), 562
error (*robot.running.model.Return attribute*), 566
error (*robot.running.model.Try attribute*), 564
error (*robot.running.model.While attribute*), 559
error() (*in module robot.api.logger*), 15
error() (*in module robot.output.librarylogger*), 306
error() (*robot.output.console.highlighting.HighlightingStream method*), 303
error() (*robot.output.console.verbose.VerboseWriter method*), 304
error() (*robot.output.filelogger.FileLogger method*), 305
error() (*robot.output.logger.Logger method*), 309
error() (*robot.output.loggerhelper.AbstractLogger method*), 309
error() (*robot.output.output.Output method*), 311
error() (*robot.utils.application.DefaultLogger method*), 591
error() (*robot.utils.importer.NoLogger method*), 602
error_occurred() (*robot.running.status.Exit method*), 583
error_occurred() (*robot.running.status.SuiteStatus method*), 583
error_occurred() (*robot.running.status.TestStatus method*), 583
ErrorDetails (*class in robot.utils.error*), 598
ErrorMessageBuilder (*class in robot.reporting.jsmodelbuilders*), 395
ErrorMessageHandler (*class in robot.result.xmlelementhandlers*), 526
ErrorReporter (*class in robot.running.builder.parsers*), 543
errors (*robot.parsing.model.blocks.Block attribute*), 340
errors (*robot.parsing.model.blocks.CommentSection attribute*), 342
errors (*robot.parsing.model.blocks.File attribute*), 341
errors (*robot.parsing.model.blocks.For attribute*), 343
errors (*robot.parsing.model.blocks.HeaderAndBody attribute*), 340
errors (*robot.parsing.model.blocks.If attribute*), 343
errors (*robot.parsing.model.blocks.Keyword attribute*), 343
errors (*robot.parsing.model.blocks.KeywordSection attribute*), 342
errors (*robot.parsing.model.blocks.Section attribute*), 341
errors (*robot.parsing.model.blocks.SettingSection attribute*), 341
errors (*robot.parsing.model.blocks.TestCase attribute*), 343

errors (*robot.parsing.model.blocks.TestCaseSection attribute*), 342
errors (*robot.parsing.model.blocks.Try attribute*), 344
errors (*robot.parsing.model.blocks.VariableSection attribute*), 342
errors (*robot.parsing.model.blocks.While attribute*), 344
errors (*robot.parsing.model.statements.Error attribute*), 387
errors (*robot.result.executionresult.Result attribute*), 413
ErrorsBuilder (*class in robot.reporting.jsmodelbuilders*), 395
ErrorSectionHeaderLexer (*class in robot.parsing.lexer.statementlexers*), 329
ErrorSectionLexer (*class in robot.parsing.lexer.blocklexers*), 321
ErrorsHandler (*class in robot.result.xmlelementhandlers*), 526
Es (*class in robot.conf.languages*), 51
escape() (*in module robot.utils.escaping*), 598
ETSource (*class in robot.utils.etreewrapper*), 598
evaluate() (*robot.libraries.BuiltIn.BuiltIn method*), 79
evaluate_expression() (*in module robot.variables.evaluation*), 607
evaluate_xpath() (*robot.libraries.XML.XML method*), 157
EvaluationNamespace (*class in robot.variables.evaluation*), 607
event_add() (*robot.libraries.dialogs_py.InputDialog method*), 174
event_add() (*robot.libraries.dialogs_py.MessageDialog method*), 160
event_add() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 202
event_add() (*robot.libraries.dialogs_py.PassFailDialog method*), 216
event_add() (*robot.libraries.dialogs_py.SelectionDialog method*), 188
event_delete() (*robot.libraries.dialogs_py.InputDialog method*), 174
event_delete() (*robot.libraries.dialogs_py.MessageDialog method*), 160
event_delete() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 202
event_delete() (*robot.libraries.dialogs_py.PassFailDialog method*), 216
event_delete() (*robot.libraries.dialogs_py.SelectionDialog method*), 188
event_generate() (*robot.libraries.dialogs_py.InputDialog method*), 174
event_generate() (*robot.libraries.dialogs_py.MessageDialog method*), 160
event_generate() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 202
event_generate() (*robot.libraries.dialogs_py.PassFailDialog method*), 216
event_generate() (*robot.libraries.dialogs_py.SelectionDialog method*), 188
event_info() (*robot.libraries.dialogs_py.InputDialog method*), 174
event_info() (*robot.libraries.dialogs_py.MessageDialog method*), 160
event_info() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 202
event_info() (*robot.libraries.dialogs_py.PassFailDialog method*), 216
event_info() (*robot.libraries.dialogs_py.SelectionDialog method*), 188
EXCEPT (*robot.model.body.BodyItem attribute*), 228
EXCEPT (*robot.model.control.Break attribute*), 249
EXCEPT (*robot.model.control.Continue attribute*), 248
EXCEPT (*robot.model.control.For attribute*), 239
EXCEPT (*robot.model.control.If attribute*), 243
EXCEPT (*robot.model.control.IfBranch attribute*), 241
EXCEPT (*robot.model.control.Return attribute*), 247
EXCEPT (*robot.model.control.Try attribute*), 245
EXCEPT (*robot.model.control.TryBranch attribute*), 244
EXCEPT (*robot.model.control.While attribute*), 240
EXCEPT (*robot.model.keyword.Keyword attribute*), 260
EXCEPT (*robot.model.message.Message attribute*), 263
EXCEPT (*robot.output.loggerhelper.Message attribute*), 309
EXCEPT (*robot.parsing.lexer.tokens.END attribute*), 338
EXCEPT (*robot.parsing.lexer.tokens.EOS attribute*), 336
EXCEPT (*robot.parsing.lexer.tokens.Token attribute*), 335
EXCEPT (*robot.result.model.Break attribute*), 488
EXCEPT (*robot.result.model.Continue attribute*), 486
EXCEPT (*robot.result.model.For attribute*), 467
EXCEPT (*robot.result.model.ForIteration attribute*), 465
EXCEPT (*robot.result.model.If attribute*), 476
EXCEPT (*robot.result.model.IfBranch attribute*), 474
EXCEPT (*robot.result.model.Keyword attribute*), 491
EXCEPT (*robot.result.model.Message attribute*), 463
EXCEPT (*robot.result.model.Return attribute*), 483
EXCEPT (*robot.result.model.Try attribute*), 481
EXCEPT (*robot.result.model.TryBranch attribute*), 479
EXCEPT (*robot.result.model.While attribute*), 472
EXCEPT (*robot.result.model.WhileIteration attribute*), 470
EXCEPT (*robot.running.model.Break attribute*), 569
EXCEPT (*robot.running.model.Continue attribute*), 567
EXCEPT (*robot.running.model.For attribute*), 558
EXCEPT (*robot.running.model.If attribute*), 562
EXCEPT (*robot.running.model.IfBranch attribute*), 560
EXCEPT (*robot.running.model.Keyword attribute*), 556
EXCEPT (*robot.running.model.Return attribute*), 566

EXCEPT (*robot.running.model.Try attribute*), 565
EXCEPT (*robot.running.model.TryBranch attribute*), 563
EXCEPT (*robot.running.model.While attribute*), 559
except_branches (*robot.model.control.Try attribute*), 245
except_branches (*robot.result.model.Try attribute*), 482
except_branches (*robot.running.model.Try attribute*), 566
ExceptHeader (*class in robot.parsing.model.statements*), 378
ExceptHeaderLexer (*class in robot.parsing.lexer.statementlexers*), 332
exclude (*robot.conf.settings.RebotSettings attribute*), 66
exclude (*robot.conf.settings.RobotSettings attribute*), 65
exclude_tags (*robot.model.filter.Filter attribute*), 254
execute() (*robot.libdoc.LibDoc method*), 619
execute() (*robot.rebot.Rebot method*), 620
execute() (*robot.run.RobotFramework method*), 622
execute() (*robot.running.timeouts.posix.Timeout method*), 549
execute() (*robot.running.timeouts.windows.Timeout method*), 549
execute() (*robot.testdoc.TestDoc method*), 624
execute() (*robot.utils.application.Application method*), 591
execute_cli() (*robot.libdoc.LibDoc method*), 619
execute_cli() (*robot.rebot.Rebot method*), 620
execute_cli() (*robot.run.RobotFramework method*), 622
execute_cli() (*robot.testdoc.TestDoc method*), 624
execute_cli() (*robot.utils.application.Application method*), 591
execute_command() (*robot.libraries.Telnet.TelnetConnection method*), 144
execute_manual_step() (*in module robot.libraries.Dialogs*), 111
ExecutionContexts (*class in robot.running.context*), 550
ExecutionErrors (*class in robot.result.executionerrors*), 412
ExecutionFailed, 615
ExecutionFailures, 615
ExecutionPassed, 616
ExecutionResult (*class in robot.libraries.Process*), 128
ExecutionResult() (*in module robot.result.resultbuilder*), 500
ExecutionResultBuilder (*class in robot.result.resultbuilder*), 500
ExecutionStatus, 614
Exit (*class in robot.running.status*), 583
exit_for_loop() (*robot.libraries.BuiltIn.BuiltIn method*), 79
exit_for_loop_if() (*robot.libraries.BuiltIn.BuiltIn method*), 80
exit_on_error (*robot.conf.settings.RobotSettings attribute*), 65
exit_on_error_message (*robot.running.status.TestMessage attribute*), 584
exit_on_failure (*robot.conf.settings.RobotSettings attribute*), 65
exit_on_failure_message (*robot.running.status.TestMessage attribute*), 584
exit_on_fatal_message (*robot.running.status.TestMessage attribute*), 584
expand_keywords (*robot.conf.settings.RebotSettings attribute*), 67
expand_keywords (*robot.reporting.jsbuildingcontext.JsBuildingContext attribute*), 394
ExpandKeywordMatcher (*class in robot.reporting.expandkeywordmatcher*), 393
expect() (*robot.libraries.Telnet.TelnetConnection method*), 145
extend() (*robot.model.body.BaseBody method*), 230
extend() (*robot.model.body.Body method*), 231
extend() (*robot.model.body.Branches method*), 233
extend() (*robot.model.itemlist.ItemList method*), 259
extend() (*robot.model.keyword.Keywords method*), 262
extend() (*robot.model.message.Messages method*), 264
extend() (*robot.model.testcase.TestCases method*), 284
extend() (*robot.model.testsuite.TestSuites method*), 288
extend() (*robot.result.model.Body method*), 458
extend() (*robot.result.model.Branches method*), 460
extend() (*robot.result.model.Iterations method*), 461
extend() (*robot.running.model.Body method*), 554
extend() (*robot.running.model.Imports method*), 576
ExtendedFinder (*class in robot.variables.finders*), 608
extension (*robot.conf.settings.RobotSettings attribute*), 65

F

FAIL (*robot.result.model.Break attribute*), 488
FAIL (*robot.result.model.Continue attribute*), 486
FAIL (*robot.result.model.For attribute*), 467

FAIL (*robot.result.model.ForIteration attribute*), 465
FAIL (*robot.result.model.If attribute*), 476
FAIL (*robot.result.model.IfBranch attribute*), 474
FAIL (*robot.result.model.Keyword attribute*), 491
FAIL (*robot.result.model.Return attribute*), 483
FAIL (*robot.result.model.StatusMixin attribute*), 464
FAIL (*robot.result.model.TestCase attribute*), 493
FAIL (*robot.result.model.TestSuite attribute*), 496
FAIL (*robot.result.model.Try attribute*), 481
FAIL (*robot.result.model.TryBranch attribute*), 479
FAIL (*robot.result.model.While attribute*), 472
FAIL (*robot.result.model.WhileIteration attribute*), 470
fail() (*in module robot.utils.asserts*), 594
fail() (*robot.libraries.BuiltIn.BuiltIn method*), 80
fail() (*robot.output.filelogger.FileLogger method*), 305
fail() (*robot.output.logger.Logger method*), 309
fail() (*robot.output.loggerhelper.AbstractLogger method*), 309
fail() (*robot.output.output.Output method*), 311
failed (*robot.model.stats.Stat attribute*), 275
failed (*robot.model.totalstatistics.TotalStatistics attribute*), 288
failed (*robot.result.model.Break attribute*), 489
failed (*robot.result.model.Continue attribute*), 487
failed (*robot.result.model.For attribute*), 468
failed (*robot.result.model.ForIteration attribute*), 466
failed (*robot.result.model.If attribute*), 477
failed (*robot.result.model.IfBranch attribute*), 475
failed (*robot.result.model.Keyword attribute*), 492
failed (*robot.result.model.Return attribute*), 484
failed (*robot.result.model.StatusMixin attribute*), 464
failed (*robot.result.model.TestCase attribute*), 494
failed (*robot.result.model.TestSuite attribute*), 496
failed (*robot.result.model.Try attribute*), 482
failed (*robot.result.model.TryBranch attribute*), 480
failed (*robot.result.model.While attribute*), 473
failed (*robot.result.model.WhileIteration attribute*), 471
failed (*robot.running.status.SuiteStatus attribute*), 583
failed (*robot.running.status.TestStatus attribute*), 583
Failure, 12
Failure (*class in robot.running.status*), 583
failure_occurred() (*robot.running.status.Exit method*), 583
failure_occurred() (*robot.running.status.SuiteStatus method*), 583
failure_occurred() (*robot.running.status.TestStatus method*), 583
false_strings (*robot.conf.languages.Bg attribute*), 60
false_strings (*robot.conf.languages.Bs attribute*), 39
false_strings (*robot.conf.languages.Cs attribute*), 36
false_strings (*robot.conf.languages.De attribute*), 43
false_strings (*robot.conf.languages.En attribute*), 35
false_strings (*robot.conf.languages.Es attribute*), 52
false_strings (*robot.conf.languages.Fi attribute*), 40
false_strings (*robot.conf.languages.Fr attribute*), 42
false_strings (*robot.conf.languages.Hi attribute*), 64
false_strings (*robot.conf.languages.It attribute*), 63
false_strings (*robot.conf.languages.Language attribute*), 33
false_strings (*robot.conf.languages.Nl attribute*), 38
false_strings (*robot.conf.languages.Pl attribute*), 49
false_strings (*robot.conf.languages.Pt attribute*), 46
false_strings (*robot.conf.languages.PtBr attribute*), 45
false_strings (*robot.conf.languages.Ro attribute*), 61
false_strings (*robot.conf.languages.Ru attribute*), 53
false_strings (*robot.conf.languages.Sv attribute*), 59
false_strings (*robot.conf.languages.Th attribute*), 47
false_strings (*robot.conf.languages.Tr attribute*), 57
false_strings (*robot.conf.languages.Uk attribute*), 50
false_strings (*robot.conf.languages.ZhCn attribute*), 54
false_strings (*robot.conf.languages.ZhTw attribute*), 56
FATAL_ERROR (*robot.parsing.lexer.tokens.END attribute*), 338
FATAL_ERROR (*robot.parsing.lexer.tokens.EOS attribute*), 336
FATAL_ERROR (*robot.parsing.lexer.tokens.Token attribute*), 335
fatal_error() (*robot.libraries.BuiltIn.BuiltIn method*), 80
FatalError, 13
fdel (*robot.utils.misc.classproperty attribute*), 604

feed() (*robot.libraries.Telnet.TerminalEmulator method*), 146
fetch_from_left() (*robot.libraries.String.String method*), 136
fetch_from_right() (*robot.libraries.String.String method*), 136
fget (*robot.utils.misc.classproperty attribute*), 604
Fi (*class in robot.conf.languages*), 39
File (*class in robot.parsing.model.blocks*), 341
file_should_be_empty() (*robot.libraries.OperatingSystem.OperatingSystem method*), 116
file_should_exist() (*robot.libraries.OperatingSystem.OperatingSystem method*), 115
file_should_not_be_empty() (*robot.libraries.OperatingSystem.OperatingSystem method*), 116
file_should_not_exist() (*robot.libraries.OperatingSystem.OperatingSystem method*), 115
FileContext (*class in robot.parsing.lexer.context*), 323
FileLexer (*class in robot.parsing.lexer.blocklexers*), 318
FileLogger (*class in robot.output.filelogger*), 305
fileno() (*robot.libraries.Telnet.TelnetConnection method*), 145
FileParser (*class in robot.parsing.parser.fileparser*), 390
FileReader (*class in robot.utils.filereader*), 598
fill_named() (*robot.running.arguments.argumentmapper.KeywordCallTemplate method*), 528
fill_positional() (*robot.running.arguments.argumentmapper.KeywordCallTemplate method*), 528
fill_rawq() (*robot.libraries.Telnet.TelnetConnection method*), 145
Filter (*class in robot.model.filter*), 254
filter() (*robot.model.body.BaseBody method*), 229
filter() (*robot.model.body.Body method*), 231
filter() (*robot.model.body.Branches method*), 233
filter() (*robot.model.testsuite.TestSuite method*), 287
filter() (*robot.output.pyloggingconf.RobotHandler method*), 312
filter() (*robot.result.model.Body method*), 458
filter() (*robot.result.model.Branches method*), 460
filter() (*robot.result.model.Iterations method*), 462
filter() (*robot.result.model.TestSuite method*), 496
filter() (*robot.running.model.Body method*), 554
filter() (*robot.running.model.TestSuite method*), 574
filter_messages() (*robot.result.model.TestSuite method*), 499
FINALLY (*robot.model.body.BodyItem attribute*), 228
FINALLY (*robot.model.control.Break attribute*), 249
FINALLY (*robot.model.control.Continue attribute*), 248
FINALLY (*robot.model.control.For attribute*), 239
FINALLY (*robot.model.control.If attribute*), 243
FINALLY (*robot.model.control.IfBranch attribute*), 241
FINALLY (*robot.model.control.Return attribute*), 247
FINALLY (*robot.model.control.Try attribute*), 246
FINALLY (*robot.model.control.TryBranch attribute*), 244
FINALLY (*robot.model.control.While attribute*), 240
FINALLY (*robot.model.keyword.Keyword attribute*), 260
FINALLY (*robot.model.message.Message attribute*), 263
FINALLY (*robot.output.loggerhelper.Message attribute*), 310
FINALLY (*robot.parsing.lexer.tokens.END attribute*), 339
FINALLY (*robot.parsing.lexer.tokens.EOS attribute*), 336
FINALLY (*robot.parsing.lexer.tokens.Token attribute*), 335
FINALLY (*robot.result.model.Break attribute*), 488
FINALLY (*robot.result.model.Continue attribute*), 486
FINALLY (*robot.result.model.For attribute*), 467
FINALLY (*robot.result.model.ForIteration attribute*), 465
FINALLY (*robot.result.model.If attribute*), 476
FINALLY (*robot.result.model.IfBranch attribute*), 474
FINALLY (*robot.result.model.Keyword attribute*), 491
FINALLY (*robot.result.model.Message attribute*), 463
FINALLY (*robot.result.model.Return attribute*), 483
FINALLY (*robot.result.model.Try attribute*), 481
FINALLY (*robot.result.model.TryBranch attribute*), 479
FINALLY (*robot.result.model.While attribute*), 472
FINALLY (*robot.result.model.WhileIteration attribute*), 470
FINALLY (*robot.running.model.Break attribute*), 569
FINALLY (*robot.running.model.Continue attribute*), 567
FINALLY (*robot.running.model.For attribute*), 558
FINALLY (*robot.running.model.If attribute*), 562
FINALLY (*robot.running.model.IfBranch attribute*), 561
FINALLY (*robot.running.model.Keyword attribute*), 556
FINALLY (*robot.running.model.Return attribute*), 566
FINALLY (*robot.running.model.Try attribute*), 565
FINALLY (*robot.running.model.TryBranch attribute*), 563
FINALLY (*robot.running.model.While attribute*), 559
finally_branch (*robot.model.control.Try attribute*), 245
finally_branch (*robot.result.model.Try attribute*), 482
finally_branch (*robot.running.model.Try attribute*), 566
FinallyHeader (*class in robot.parsing.model.statements*), 379

FinallyHeaderLexer (*class in robot.parsing.lexer.statementlexers*), 332
find() (*robot.utils.recommendations.RecommendationFinder method*), 605
find() (*robot.variables.finders.EmptyFinder method*), 608
find() (*robot.variables.finders.EnvironmentFinder method*), 608
find() (*robot.variables.finders.ExtendedFinder method*), 608
find() (*robot.variables.finders.InlinePythonFinder method*), 608
find() (*robot.variables.finders.NumberFinder method*), 608
find() (*robot.variables.finders.StoredFinder method*), 608
find() (*robot.variables.finders.VariableFinder method*), 608
find_all() (*robot.libraries.XML.ElementFinder method*), 157
find_and_format() (*robot.utils.recommendations.RecommendationFinder method*), 605
find_from() (*robot.parsing.model.blocks.FirstStatementFinder class method*), 345
find_from() (*robot.parsing.model.blocks.LastStatementFinder class method*), 345
FirstStatementFinder (*class in robot.parsing.model.blocks*), 345
Fixture (*class in robot.parsing.model.statements*), 349
fixture_class (*robot.model.testcase.TestCase attribute*), 282
fixture_class (*robot.model.testsuite.TestSuite attribute*), 285
fixture_class (*robot.result.model.TestCase attribute*), 493
fixture_class (*robot.result.model.TestSuite attribute*), 496
fixture_class (*robot.running.model.TestCase attribute*), 570
fixture_class (*robot.running.model.TestSuite attribute*), 572
flatten() (*robot.model.body.BaseBody method*), 230
flatten() (*robot.model.body.Body method*), 231
flatten() (*robot.model.body.Branches method*), 233
flatten() (*robot.result.model.Body method*), 458
flatten() (*robot.result.model.Branches method*), 460
flatten() (*robot.result.model.Iterations method*), 462
flatten() (*robot.running.model.Body method*), 554
flatten_keywords (*robot.conf.settings.RebotSettings attribute*), 66
flatten_keywords (*robot.conf.settings.RobotSettings attribute*), 65
FlattenByNameMatcher (*class in robot.result.flattenkeywordmatcher*), 415
FlattenByTagMatcher (*class in robot.result.flattenkeywordmatcher*), 415
FlattenByTypeMatcher (*class in robot.result.flattenkeywordmatcher*), 415
flavor (*robot.model.control.For attribute*), 238
flavor (*robot.parsing.model.blocks.For attribute*), 344
flavor (*robot.parsing.model.statements.ForHeader attribute*), 372
flavor (*robot.result.model.For attribute*), 468
flavor (*robot.running.bodyrunner.ForInEnumerateRunner attribute*), 549
flavor (*robot.running.bodyrunner.ForInRangeRunner attribute*), 549
flavor (*robot.running.bodyrunner.ForInRunner attribute*), 549
flavor (*robot.running.bodyrunner.ForInZipRunner attribute*), 549
flavor (*robot.running.model.For attribute*), 558
FloatConverter (*class in robot.running.arguments.typeconverters*), 533
flush() (*robot.output.console.highlighting.HighlightingStream method*), 303
flush() (*robot.output.pyloggingconf.RobotHandler method*), 312
focus() (*robot.libraries.dialogs_py.InputDialog method*), 174
focus() (*robot.libraries.dialogs_py.MessageDialog method*), 160
focus() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 202
focus() (*robot.libraries.dialogs_py.PassFailDialog method*), 216
focus() (*robot.libraries.dialogs_py.SelectionDialog method*), 188
focus_displayof() (*robot.libraries.dialogs_py.InputDialog method*), 174
focus_displayof() (*robot.libraries.dialogs_py.MessageDialog method*), 160
focus_displayof() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 202
focus_displayof() (*robot.libraries.dialogs_py.PassFailDialog method*), 216
focus_displayof() (*robot.libraries.dialogs_py.SelectionDialog method*), 188
focus_force() (*robot.libraries.dialogs_py.InputDialog method*), 174
focus_force() (*robot.libraries.dialogs_py.MessageDialog*

method), 160
focus_force() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 202
focus_force() (*robot.libraries.dialogs_py.PassFailDialog method*), 216
focus_force() (*robot.libraries.dialogs_py.SelectionDialog method*), 188
focus_get() (*robot.libraries.dialogs_py.InputDialog method*), 175
focus_get() (*robot.libraries.dialogs_py.MessageDialog method*), 161
focus_get() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 203
focus_get() (*robot.libraries.dialogs_py.PassFailDialog method*), 217
focus_get() (*robot.libraries.dialogs_py.SelectionDialog method*), 189
focus_lastfor() (*robot.libraries.dialogs_py.InputDialog method*), 175
focus_lastfor() (*robot.libraries.dialogs_py.MessageDialog method*), 161
focus_lastfor() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 203
focus_lastfor() (*robot.libraries.dialogs_py.PassFailDialog method*), 217
focus_lastfor() (*robot.libraries.dialogs_py.SelectionDialog method*), 189
focus_set() (*robot.libraries.dialogs_py.InputDialog method*), 175
focus_set() (*robot.libraries.dialogs_py.MessageDialog method*), 161
focus_set() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 203
focus_set() (*robot.libraries.dialogs_py.PassFailDialog method*), 217
focus_set() (*robot.libraries.dialogs_py.SelectionDialog method*), 189
focusmodel() (*robot.libraries.dialogs_py.InputDialog method*), 175
focusmodel() (*robot.libraries.dialogs_py.MessageDialog method*), 161
focusmodel() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 203
focusmodel() (*robot.libraries.dialogs_py.PassFailDialog method*), 217
focusmodel() (*robot.libraries.dialogs_py.SelectionDialog method*), 189
For (*class in robot.model.control*), 238
For (*class in robot.parsing.model.blocks*), 343
For (*class in robot.result.model*), 467
For (*class in robot.running.model*), 557
FOR (*robot.model.body.BodyItem attribute*), 228
FOR (*robot.model.control.Break attribute*), 249
FOR (*robot.model.control.Continue attribute*), 248
FOR (*robot.model.control.For attribute*), 239
FOR (*robot.model.control.If attribute*), 243
FOR (*robot.model.control.IfBranch attribute*), 241
FOR (*robot.model.control.Return attribute*), 247
FOR (*robot.model.control.Try attribute*), 246
FOR (*robot.model.control.TryBranch attribute*), 244
FOR (*robot.model.control.While attribute*), 240
FOR (*robot.model.keyword.Keyword attribute*), 260
FOR (*robot.model.message.Message attribute*), 263
FOR (*robot.output.loggerhelper.Message attribute*), 310
FOR (*robot.parsing.lexer.tokens.END attribute*), 339
FOR (*robot.parsing.lexer.tokens.EOS attribute*), 336
FOR (*robot.parsing.lexer.tokens.Token attribute*), 334
FOR (*robot.result.model.Break attribute*), 488
FOR (*robot.result.model.Continue attribute*), 486
FOR (*robot.result.model.For attribute*), 467
FOR (*robot.result.model.ForIteration attribute*), 465
FOR (*robot.result.model.If attribute*), 476
FOR (*robot.result.model.IfBranch attribute*), 474
FOR (*robot.result.model.Keyword attribute*), 491
FOR (*robot.result.model.Message attribute*), 463
FOR (*robot.result.model.Return attribute*), 483
FOR (*robot.result.model.Try attribute*), 481
FOR (*robot.result.model.TryBranch attribute*), 479
FOR (*robot.result.model.While attribute*), 472
FOR (*robot.result.model.WhileIteration attribute*), 470
FOR (*robot.running.model.Break attribute*), 569
FOR (*robot.running.model.Continue attribute*), 567
FOR (*robot.running.model.For attribute*), 558
FOR (*robot.running.model.If attribute*), 562
FOR (*robot.running.model.IfBranch attribute*), 561
FOR (*robot.running.model.Keyword attribute*), 556
FOR (*robot.running.model.Return attribute*), 566
FOR (*robot.running.model.Try attribute*), 565
FOR (*robot.running.model.TryBranch attribute*), 563
FOR (*robot.running.model.While attribute*), 559
for_class (*robot.model.body.BaseBody attribute*), 229
for_class (*robot.model.body.Body attribute*), 231
for_class (*robot.model.body.Branches attribute*), 233
for_class (*robot.result.model.Body attribute*), 459
for_class (*robot.result.model.Branches attribute*), 460
for_class (*robot.result.model.Iterations attribute*), 462
for_class (*robot.running.model.Body attribute*), 554
for_converter() (*robot.running.arguments.customconverters.Convert class method*), 531
for_enum() (*robot.libdocpkg.datatypes.TypeDoc class method*), 70
FOR_SEPARATOR (*robot.parsing.lexer.tokens.END attribute*), 339
FOR_SEPARATOR (*robot.parsing.lexer.tokens.EOS attribute*), 336

FOR_SEPARATOR (*robot.parsing.lexer.tokens.Token attribute*), 334
for_type() (*robot.libdocpkg.datatypes.TypeDoc class method*), 70
for_typed_dict() (*robot.libdocpkg.datatypes.TypeDoc class method*), 70
ForBuilder (*class in robot.running.builder.transformers*), 546
FORCE_TAGS (*robot.parsing.lexer.tokens.END attribute*), 339
FORCE_TAGS (*robot.parsing.lexer.tokens.EOS attribute*), 336
FORCE_TAGS (*robot.parsing.lexer.tokens.Token attribute*), 334
force_tags (*robot.running.builder.settings.Defaults attribute*), 543
ForceTags (*class in robot.parsing.model.statements*), 354
forget() (*robot.libraries.dialogs_py.InputDialog method*), 175
forget() (*robot.libraries.dialogs_py.MessageDialog method*), 161
forget() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 203
forget() (*robot.libraries.dialogs_py.PassFailDialog method*), 217
forget() (*robot.libraries.dialogs_py.SelectionDialog method*), 189
ForHandler (*class in robot.result.xmlelementhandlers*), 520
ForHeader (*class in robot.parsing.model.statements*), 372
ForHeaderLexer (*class in robot.parsing.lexer.statementlexers*), 331
ForInEnumerateRunner (*class in robot.running.bodyrunner*), 549
ForInRangeRunner (*class in robot.running.bodyrunner*), 549
ForInRunner (*class in robot.running.bodyrunner*), 549
ForInZipRunner (*class in robot.running.bodyrunner*), 549
ForIteration (*class in robot.result.model*), 464
ForLexer (*class in robot.parsing.lexer.blocklexers*), 322
ForLoopItemsRemover (*class in robot.result.keywordremover*), 432
format() (*robot.output.pyloggingconf.RobotHandler method*), 312
format() (*robot.utils.htmlformatters.HeaderFormatter method*), 599
format() (*robot.utils.htmlformatters.HtmlFormatter method*), 599
format() (*robot.utils.htmlformatters.LineFormatter method*), 599
format() (*robot.utils.htmlformatters.ListFormatter method*), 600
format() (*robot.utils.htmlformatters.ParagraphFormatter method*), 600
format() (*robot.utils.htmlformatters.PreformattedFormatter method*), 600
format() (*robot.utils.htmlformatters.RulerFormatter method*), 599
format() (*robot.utils.htmlformatters.TableFormatter method*), 600
format() (*robot.utils.recommendations.RecommendationFinder method*), 605
format_error() (*in module robot.running.builder.transformers*), 548
format_line() (*robot.utils.htmlformatters.HeaderFormatter method*), 599
format_line() (*robot.utils.htmlformatters.RulerFormatter method*), 599
format_link() (*robot.utils.htmlformatters.LinkFormatter method*), 599
format_name() (*in module robot.running.builder.parsers*), 543
format_recommendations() (*robot.running.namespace.KeywordRecommendationFinder static method*), 578
format_string() (*robot.libraries.String.String method*), 133
format_url() (*robot.utils.htmlformatters.LinkFormatter method*), 599
ForParser (*class in robot.parsing.parser.blockparsers*), 390
ForRunner() (*in module robot.running.bodyrunner*), 549
Fr (*class in robot.conf.languages*), 41
frame() (*robot.libraries.dialogs_py.InputDialog method*), 175
frame() (*robot.libraries.dialogs_py.MessageDialog method*), 161
frame() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 203
frame() (*robot.libraries.dialogs_py.PassFailDialog method*), 217
frame() (*robot.libraries.dialogs_py.SelectionDialog method*), 189
FrameworkError, 613
frange() (*in module robot.utils.frange*), 599
from_bytes() (*robot.reporting.stringcache.StringIndex method*), 401
from_dict() (*robot.running.arguments.customconverters.CustomArgum class method*), 531
from_file_system() (*robot.running.model.TestSuite class method*), 572

from_model() (*robot.running.model.TestSuite class method*), 572
from_name() (*robot.conf.languages.Bg class method*), 60
from_name() (*robot.conf.languages.Bs class method*), 39
from_name() (*robot.conf.languages.Cs class method*), 36
from_name() (*robot.conf.languages.De class method*), 43
from_name() (*robot.conf.languages.En class method*), 35
from_name() (*robot.conf.languages.Es class method*), 52
from_name() (*robot.conf.languages.Fi class method*), 41
from_name() (*robot.conf.languages.Fr class method*), 42
from_name() (*robot.conf.languages.Hi class method*), 64
from_name() (*robot.conf.languages.It class method*), 63
from_name() (*robot.conf.languages.Language class method*), 33
from_name() (*robot.conf.languages.Nl class method*), 38
from_name() (*robot.conf.languages.Pl class method*), 49
from_name() (*robot.conf.languages.Pt class method*), 46
from_name() (*robot.conf.languages.PtBr class method*), 45
from_name() (*robot.conf.languages.Ro class method*), 61
from_name() (*robot.conf.languages.Ru class method*), 53
from_name() (*robot.conf.languages.Sv class method*), 59
from_name() (*robot.conf.languages.Th class method*), 47
from_name() (*robot.conf.languages.Tr class method*), 57
from_name() (*robot.conf.languages.Uk class method*), 50
from_name() (*robot.conf.languages.ZhCn class method*), 54
from_name() (*robot.conf.languages.ZhTw class method*), 56
from_name() (*robot.running.arguments.embedded.EmbeddedArguments class method*), 531
from_params() (*robot.parsing.model.statements.Arguments class method*), 368
from_params() (*robot.parsing.model.statements.Break class method*), 384
from_params() (*robot.parsing.model.statements.Comment class method*), 385
from_params() (*robot.parsing.model.statements.Config class method*), 386
from_params() (*robot.parsing.model.statements.Continue class method*), 384
from_params() (*robot.parsing.model.statements.DefaultTags class method*), 355
from_params() (*robot.parsing.model.statements.Documentation class method*), 353
from_params() (*robot.parsing.model.statements.DocumentationOrMeta class method*), 346
from_params() (*robot.parsing.model.statements.ElseHeader class method*), 376
from_params() (*robot.parsing.model.statements.ElseIfHeader class method*), 375
from_params() (*robot.parsing.model.statements.EmptyLine class method*), 388
from_params() (*robot.parsing.model.statements.End class method*), 380
from_params() (*robot.parsing.model.statements.Error class method*), 387
from_params() (*robot.parsing.model.statements.ExceptHeader class method*), 378
from_params() (*robot.parsing.model.statements.FinallyHeader class method*), 379
from_params() (*robot.parsing.model.statements.Fixture class method*), 349
from_params() (*robot.parsing.model.statements.ForceTags class method*), 355
from_params() (*robot.parsing.model.statements.ForHeader class method*), 372
from_params() (*robot.parsing.model.statements.IfElseHeader class method*), 373
from_params() (*robot.parsing.model.statements.IfHeader class method*), 373
from_params() (*robot.parsing.model.statements.InlineIfHeader class method*), 374
from_params() (*robot.parsing.model.statements.KeywordCall class method*), 370
from_params() (*robot.parsing.model.statements.KeywordName class method*), 363
from_params() (*robot.parsing.model.statements.KeywordTags class method*), 356
from_params() (*robot.parsing.model.statements.LibraryImport class method*), 350
from_params() (*robot.parsing.model.statements.LoopControl class method*), 383
from_params() (*robot.parsing.model.statements.Metadata class method*), 354
from_params() (*robot.parsing.model.statements.MultiValue class method*), 348
from_params() (*robot.parsing.model.statements.NoArgumentHeader class method*), 377

from_params() (*robot.parsing.model.statements.ResourceImport class method*), 351
from_params() (*robot.parsing.model.statements.Return class method*), 369
from_params() (*robot.parsing.model.statements.ReturnStatement class method*), 382
from_params() (*robot.parsing.model.statements.SectionHeader class method*), 350
from_params() (*robot.parsing.model.statements.Setup class method*), 364
from_params() (*robot.parsing.model.statements.SingleValue class method*), 347
from_params() (*robot.parsing.model.statements.Statement class method*), 346
from_params() (*robot.parsing.model.statements.SuiteSetup class method*), 357
from_params() (*robot.parsing.model.statements.SuiteTeardown class method*), 358
from_params() (*robot.parsing.model.statements.Tags class method*), 366
from_params() (*robot.parsing.model.statements.Teardown class method*), 365
from_params() (*robot.parsing.model.statements.Template class method*), 367
from_params() (*robot.parsing.model.statements.TemplateArguments class method*), 371
from_params() (*robot.parsing.model.statements.TestCaseName class method*), 363
from_params() (*robot.parsing.model.statements.TestSetup class method*), 359
from_params() (*robot.parsing.model.statements.TestTeardown class method*), 360
from_params() (*robot.parsing.model.statements.TestTemplate class method*), 360
from_params() (*robot.parsing.model.statements.TestTimeout class method*), 361
from_params() (*robot.parsing.model.statements.Timeout class method*), 368
from_params() (*robot.parsing.model.statements.TryHeader class method*), 378
from_params() (*robot.parsing.model.statements.Variable class method*), 362
from_params() (*robot.parsing.model.statements.VariablesImport class method*), 352
from_params() (*robot.parsing.model.statements.WhileHeader class method*), 381
from_token() (*robot.parsing.lexer.tokens.END class method*), 338
from_token() (*robot.parsing.lexer.tokens.EOS class method*), 336
from_tokens() (*robot.parsing.model.statements.Arguments class method*), 369
from_tokens() (*robot.parsing.model.statements.Break class method*), 385
from_tokens() (*robot.parsing.model.statements.Comment class method*), 385
from_tokens() (*robot.parsing.model.statements.Config class method*), 386
from_tokens() (*robot.parsing.model.statements.Continue class method*), 384
from_tokens() (*robot.parsing.model.statements.DefaultTags class method*), 356
from_tokens() (*robot.parsing.model.statements.Documentation class method*), 353
from_tokens() (*robot.parsing.model.statements.DocumentationOrMeta class method*), 347
from_tokens() (*robot.parsing.model.statements.ElseHeader class method*), 376
from_tokens() (*robot.parsing.model.statements.ElseIfHeader class method*), 375
from_tokens() (*robot.parsing.model.statements.EmptyLine class method*), 388
from_tokens() (*robot.parsing.model.statements.End class method*), 380
from_tokens() (*robot.parsing.model.statements.Error class method*), 387
from_tokens() (*robot.parsing.model.statements.ExceptHeader class method*), 379
from_tokens() (*robot.parsing.model.statements.FinallyHeader class method*), 380
from_tokens() (*robot.parsing.model.statements.Fixture class method*), 349
from_tokens() (*robot.parsing.model.statements.ForceTags class method*), 355
from_tokens() (*robot.parsing.model.statements.ForHeader class method*), 372
from_tokens() (*robot.parsing.model.statements.IfElseHeader class method*), 373
from_tokens() (*robot.parsing.model.statements.IfHeader class method*), 374
from_tokens() (*robot.parsing.model.statements.InlineIfHeader class method*), 375
from_tokens() (*robot.parsing.model.statements.KeywordCall class method*), 370
from_tokens() (*robot.parsing.model.statements.KeywordName class method*), 364
from_tokens() (*robot.parsing.model.statements.KeywordTags class method*), 356
from_tokens() (*robot.parsing.model.statements.LibraryImport class method*), 351
from_tokens() (*robot.parsing.model.statements.LoopControl class method*), 383
from_tokens() (*robot.parsing.model.statements.Metadata class method*), 354
from_tokens() (*robot.parsing.model.statements.MultiValue class method*), 348
from_tokens() (*robot.parsing.model.statements.NoArgumentHeader class method*), 377

from_tokens() (*robot.parsing.model.statements.ResourceImport class method*), 352
from_tokens() (*robot.parsing.model.statements.Return class method*), 370
from_tokens() (*robot.parsing.model.statements.ReturnStatement class method*), 382
from_tokens() (*robot.parsing.model.statements.SectionHeader class method*), 350
from_tokens() (*robot.parsing.model.statements.Setup class method*), 365
from_tokens() (*robot.parsing.model.statements.SingleValue class method*), 347
from_tokens() (*robot.parsing.model.statements.Statement class method*), 346
from_tokens() (*robot.parsing.model.statements.SuiteSetup class method*), 357
from_tokens() (*robot.parsing.model.statements.SuiteTeardown class method*), 358
from_tokens() (*robot.parsing.model.statements.Tags class method*), 366
from_tokens() (*robot.parsing.model.statements.Teardown class method*), 365
from_tokens() (*robot.parsing.model.statements.Template class method*), 367
from_tokens() (*robot.parsing.model.statements.TemplateArguments class method*), 371
from_tokens() (*robot.parsing.model.statements.TestCaseName class method*), 363
from_tokens() (*robot.parsing.model.statements.TestSetup class method*), 359
from_tokens() (*robot.parsing.model.statements.TestTeardown class method*), 360
from_tokens() (*robot.parsing.model.statements.TestTemplate class method*), 361
from_tokens() (*robot.parsing.model.statements.TestTimeout class method*), 361
from_tokens() (*robot.parsing.model.statements.Timeout class method*), 368
from_tokens() (*robot.parsing.model.statements.TryHeader class method*), 378
from_tokens() (*robot.parsing.model.statements.Variable class method*), 362
from_tokens() (*robot.parsing.model.statements.VariablesImport class method*), 352
from_tokens() (*robot.parsing.model.statements.WhileHeader class method*), 381
fromkeys() (*robot.utils.dotdict.DotDict method*), 596
FrozenSetConverter (*class in robot.running.arguments.typeconverters*), 539
fset (*robot.utils.misc.classproperty attribute*), 604
full_message (*robot.result.model.TestSuite attribute*), 499
full_name() (*in module robot.model.modelobject*), 265

G

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*), 136
GeneratorWriter (*class in robot.htmldata.htmlfilewriter*), 67
generic_visit() (*robot.parsing.model.blocks.FirstStatementFinder method*), 345
generic_visit() (*robot.parsing.model.blocks.LastStatementFinder method*), 345
generic_visit() (*robot.parsing.model.blocks.ModelValidator method*), 345
generic_visit() (*robot.parsing.model.blocks.ModelWriter method*), 345
generic_visit() (*robot.parsing.model.visitor.ModelTransformer method*), 389
generic_visit() (*robot.parsing.model.visitor.ModelVisitor method*), 389
generic_visit() (*robot.parsing.parser.parser.SetLanguages method*), 392
generic_visit() (*robot.running.builder.parsers.ErrorReporter method*), 543
generic_visit() (*robot.running.builder.transformers.ForBuilder method*), 546
generic_visit() (*robot.running.builder.transformers.IfBuilder method*), 547
generic_visit() (*robot.running.builder.transformers.KeywordBuilder method*), 546
generic_visit() (*robot.running.builder.transformers.ResourceBuilder method*), 545
generic_visit() (*robot.running.builder.transformers.SettingsBuilder method*), 544
generic_visit() (*robot.running.builder.transformers.SuiteBuilder method*), 544
generic_visit() (*robot.running.builder.transformers.TestCaseBuilder method*), 545
generic_visit() (*robot.running.builder.transformers.TryBuilder method*), 547
generic_visit() (*robot.running.builder.transformers.WhileBuilder method*), 548
geometry() (*robot.libraries.dialogs_py.InputDialog method*), 175
geometry() (*robot.libraries.dialogs_py.MessageDialog method*), 161

geometry() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 203
geometry() (*robot.libraries.dialogs_py.PassFailDialog method*), 217
geometry() (*robot.libraries.dialogs_py.SelectionDialog method*), 189
get() (*robot.model.metadata.Metadata method*), 264
get() (*robot.utils.dotdict.DotDict method*), 596
get() (*robot.utils.normalizing.NormalizedDict method*), 605
get() (*robot.variables.evaluation.EvaluationNamespace method*), 607
get() (*robot.variables.store.VariableStore method*), 611
get_arguments() (*robot.output.listenerarguments.EndKeywordArguments method*), 306
get_arguments() (*robot.output.listenerarguments.EndSuiteArguments method*), 306
get_arguments() (*robot.output.listenerarguments.EndTestArguments method*), 306
get_arguments() (*robot.output.listenerarguments.ListenerArguments method*), 306
get_arguments() (*robot.output.listenerarguments.MessageArguments method*), 306
get_arguments() (*robot.output.listenerarguments.StartKeywordArguments method*), 306
get_arguments() (*robot.output.listenerarguments.StartSuiteArguments method*), 306
get_arguments() (*robot.output.listenerarguments.StartTestArguments method*), 306
get_attributes() (*robot.model.stats.CombinedTagStat method*), 276
get_attributes() (*robot.model.stats.Stat method*), 275
get_attributes() (*robot.model.stats.SuiteStat method*), 275
get_attributes() (*robot.model.stats.TagStat method*), 276
get_attributes() (*robot.model.stats.TotalStat method*), 275
get_binary_file() (*robot.libraries.OperatingSystem.OperatingSystem method*), 114
get_char_width() (*in module robot.utils.charwidth*), 595
get_child_elements() (*robot.libraries.XML.XML method*), 151
get_child_handler() (*robot.result.xmlelementhandlers.ArgumentHandler method*), 526
get_child_handler() (*robot.result.xmlelementhandlers.ArgumentsHandler method*), 525
get_child_handler() (*robot.result.xmlelementhandlers.AssignHandler method*), 525
get_child_handler() (*robot.result.xmlelementhandlers.BranchHandler method*), 521
get_child_handler() (*robot.result.xmlelementhandlers.BreakHandler method*), 522
get_child_handler() (*robot.result.xmlelementhandlers.ContinueHandler method*), 522
get_child_handler() (*robot.result.xmlelementhandlers.DocHandler method*), 523
get_child_handler() (*robot.result.xmlelementhandlers.ElementHandler method*), 519
get_child_handler() (*robot.result.xmlelementhandlers.ErrorMessageHandler method*), 526
get_child_handler() (*robot.result.xmlelementhandlers.ErrorsHandler method*), 526
get_child_handler() (*robot.result.xmlelementhandlers.ForHandler method*), 520
get_child_handler() (*robot.result.xmlelementhandlers.IfHandler method*), 521
get_child_handler() (*robot.result.xmlelementhandlers.IterationHandler method*), 521
get_child_handler() (*robot.result.xmlelementhandlers.KeywordHandler method*), 520
get_child_handler() (*robot.result.xmlelementhandlers.MessageHandler method*), 523
get_child_handler() (*robot.result.xmlelementhandlers.MetadataHandler method*), 523
get_child_handler() (*robot.result.xmlelementhandlers.MetadataItemHandler method*), 524
get_child_handler() (*robot.result.xmlelementhandlers.MetaHandler method*), 524
get_child_handler() (*robot.result.xmlelementhandlers.PatternHandler method*), 522
get_child_handler() (*robot.result.xmlelementhandlers.ReturnHandler method*), 522
get_child_handler() (*robot.result.xmlelementhandlers.RobotHandler*

method), 519
get_child_handler() (*robot.result.xmlelementhandlers.RootHandler method*), 519
get_child_handler() (*robot.result.xmlelementhandlers.StatisticsHandler method*), 526
get_child_handler() (*robot.result.xmlelementhandlers.StatusHandler method*), 523
get_child_handler() (*robot.result.xmlelementhandlers.SuiteHandler method*), 519
get_child_handler() (*robot.result.xmlelementhandlers.TagHandler method*), 524
get_child_handler() (*robot.result.xmlelementhandlers.TagsHandler method*), 524
get_child_handler() (*robot.result.xmlelementhandlers.TestHandler method*), 520
get_child_handler() (*robot.result.xmlelementhandlers.TimeoutHandler method*), 525
get_child_handler() (*robot.result.xmlelementhandlers.TryHandler method*), 521
get_child_handler() (*robot.result.xmlelementhandlers.ValueHandler method*), 526
get_child_handler() (*robot.result.xmlelementhandlers.VarHandler method*), 525
get_child_handler() (*robot.result.xmlelementhandlers.WhileHandler method*), 521
get_combined_stats() (*robot.model.tagstatistics.TagStatInfo method*), 282
get_command() (*robot.libraries.Process.ProcessConfiguration method*), 128
get_connection() (*robot.utils.connectioncache.ConnectionCache method*), 596
get_console_encoding() (*in module robot.utils.encodingsniffer*), 597
get_converter_info() (*robot.running.arguments.customconverters.CustomArgumentConverters method*), 531
get_count() (*robot.libraries.BuiltIn.BuiltIn method*), 80
get_current_date() (*in module robot.libraries.DateTime*), 109
get_dictionary_items() (*robot.libraries.Collections.Collections method*), 103
get_dictionary_keys() (*robot.libraries.Collections.Collections method*), 103
get_dictionary_values() (*robot.libraries.Collections.Collections method*), 103
get_doc() (*robot.model.tagstatistics.TagStatInfo method*), 282
get_element() (*robot.libraries.XML.XML method*), 151
get_element_attribute() (*robot.libraries.XML.XML method*), 153
get_element_attributes() (*robot.libraries.XML.XML method*), 153
get_element_count() (*robot.libraries.XML.XML method*), 151
get_element_text() (*robot.libraries.XML.XML method*), 152
get_elements() (*robot.libraries.XML.XML method*), 151
get_elements_texts() (*robot.libraries.XML.XML method*), 152
get_environment_variable() (*robot.libraries.OperatingSystem.OperatingSystem method*), 118
get_environment_variables() (*robot.libraries.OperatingSystem.OperatingSystem method*), 119
get_error_details() (*in module robot.utils.error*), 598
get_error_message() (*in module robot.utils.error*), 598
get_errors() (*robot.errors.BreakLoop method*), 617
get_errors() (*robot.errors.ContinueLoop method*), 617
get_errors() (*robot.errors.ExecutionFailed method*), 615
get_errors() (*robot.errors.ExecutionFailures method*), 615
get_errors() (*robot.errors.ExecutionPassed method*), 616
get_errors() (*robot.errors.ExecutionStatus method*), 615
get_errors() (*robot.errors.HandlerExecutionFailed method*), 615
get_errors() (*robot.errors.PassExecution method*), 617
get_errors() (*robot.errors.ReturnFromKeyword method*), 618
get_errors() (*robot.errors.UserKeywordExecutionFailed method*), 616
get_file() (*robot.libraries.OperatingSystem.OperatingSystem*

method), 114
get_file_size() (*robot.libraries.OperatingSystem.OperatingSystem method*), 121
get_from_dictionary() (*robot.libraries.Collections.Collections method*), 103
get_from_list() (*robot.libraries.Collections.Collections method*), 104
get_full_version() (*in module robot.version*), 625
get_generation_time() (*in module robot.libdocpkg.output*), 72
get_handlers() (*robot.running.handlerstore.HandlerStore method*), 551
get_host_info() (*robot.libraries.Remote.TimeoutHTTPSTransport method*), 129
get_host_info() (*robot.libraries.Remote.TimeoutHTTPTransport method*), 129
get_index_from_list() (*robot.libraries.Collections.Collections method*), 104
get_init_model() (*in module robot.parsing.parser.parser*), 392
get_init_tokens() (*in module robot.parsing.lexer.lexer*), 325
get_interpreter() (*in module robot.version*), 625
get_keyword_arguments() (*robot.libraries.Remote.Remote method*), 128
get_keyword_arguments() (*robot.libraries.Remote.XmlRpcRemoteClient method*), 129
get_keyword_documentation() (*robot.libraries.Remote.Remote method*), 128
get_keyword_documentation() (*robot.libraries.Remote.XmlRpcRemoteClient method*), 129
get_keyword_names() (*robot.libraries.Remote.Remote method*), 128
get_keyword_names() (*robot.libraries.Remote.XmlRpcRemoteClient method*), 129
get_keyword_names() (*robot.libraries.Telnet.Telnet method*), 140
get_keyword_tags() (*robot.libraries.Remote.Remote method*), 128
get_keyword_tags() (*robot.libraries.Remote.XmlRpcRemoteClient method*), 129
get_keyword_types() (*robot.libraries.Remote.Remote method*), 128
get_keyword_types() (*robot.libraries.Remote.XmlRpcRemoteClient method*), 129
get_length() (*robot.libraries.BuiltIn.BuiltIn method*), 80
get_library() (*robot.running.namespace.KeywordStore method*), 578
get_library_information() (*robot.libraries.Remote.XmlRpcRemoteClient method*), 129
get_library_instance() (*robot.libraries.BuiltIn.BuiltIn method*), 81
get_library_instance() (*robot.running.namespace.Namespace method*), 578
get_library_instances() (*robot.running.namespace.Namespace method*), 578
get_line() (*robot.libraries.String.String method*), 133
get_line_count() (*robot.libraries.String.String method*), 133
get_lines_containing_string() (*robot.libraries.String.String method*), 133
get_lines_matching_pattern() (*robot.libraries.String.String method*), 134
get_lines_matching_regexp() (*robot.libraries.String.String method*), 134
get_link() (*robot.model.tagstatistics.TagStatLink method*), 282
get_links() (*robot.model.tagstatistics.TagStatInfo method*), 282
get_match_count() (*robot.libraries.Collections.Collections method*), 101
get_matches() (*robot.libraries.Collections.Collections method*), 101
get_message() (*robot.running.timeouts.KeywordTimeout method*), 548
get_message() (*robot.running.timeouts.TestTimeout method*), 548
get_model() (*in module robot.parsing.parser.parser*), 391
get_modified_time() (*robot.libraries.OperatingSystem.OperatingSystem method*), 121
get_name() (*robot.output.pyloggingconf.RobotHandler method*), 312
get_process_id() (*robot.libraries.Process.Process method*), 127
get_process_object() (*robot.libraries.Process.Process method*),

127
get_process_result() (*robot.libraries.Process.Process method*), 127
get_rebot_settings() (*robot.conf.settings.RobotSettings method*), 65
get_regexp_matches() (*robot.libraries.String.String method*), 134
get_resource_model() (*in module robot.parsing.parser.parser*), 392
get_resource_tokens() (*in module robot.parsing.lexer.lexer*), 325
get_runner() (*robot.running.namespace.KeywordStore method*), 578
get_runner() (*robot.running.namespace.Namespace method*), 578
get_selection_from_user() (*in module robot.libraries.Dialogs*), 112
get_selections_from_user() (*in module robot.libraries.Dialogs*), 112
get_shortdoc_from_html() (*robot.libdocpkg.htmlutils.HtmlToText method*), 70
get_slice_from_list() (*robot.libraries.Collections.Collections method*), 104
get_socket() (*robot.libraries.Telnet.TelnetConnection method*), 145
get_stat() (*robot.model.tagstatistics.TagStatInfo method*), 282
get_substring() (*robot.libraries.String.String method*), 136
get_system_encoding() (*in module robot.utils.encodingsniffer*), 597
get_time() (*robot.libraries.BuiltIn.BuiltIn method*), 81
get_token() (*robot.parsing.model.statements.Arguments method*), 369
get_token() (*robot.parsing.model.statements.Break method*), 385
get_token() (*robot.parsing.model.statements.Comment method*), 385
get_token() (*robot.parsing.model.statements.Config method*), 386
get_token() (*robot.parsing.model.statements.Continue method*), 384
get_token() (*robot.parsing.model.statements.DefaultTags method*), 356
get_token() (*robot.parsing.model.statements.Documentation method*), 353
get_token() (*robot.parsing.model.statements.DocumentationOrMetadata method*), 347
get_token() (*robot.parsing.model.statements.ElseHeader method*), 376
get_token() (*robot.parsing.model.statements.ElseIfHeader method*), 375
get_token() (*robot.parsing.model.statements.EmptyLine method*), 388
get_token() (*robot.parsing.model.statements.End method*), 380
get_token() (*robot.parsing.model.statements.Error method*), 387
get_token() (*robot.parsing.model.statements.ExceptHeader method*), 379
get_token() (*robot.parsing.model.statements.FinallyHeader method*), 380
get_token() (*robot.parsing.model.statements.Fixture method*), 349
get_token() (*robot.parsing.model.statements.ForceTags method*), 355
get_token() (*robot.parsing.model.statements.ForHeader method*), 372
get_token() (*robot.parsing.model.statements.IfElseHeader method*), 373
get_token() (*robot.parsing.model.statements.IfHeader method*), 374
get_token() (*robot.parsing.model.statements.InlineIfHeader method*), 375
get_token() (*robot.parsing.model.statements.KeywordCall method*), 370
get_token() (*robot.parsing.model.statements.KeywordName method*), 364
get_token() (*robot.parsing.model.statements.KeywordTags method*), 356
get_token() (*robot.parsing.model.statements.LibraryImport method*), 351
get_token() (*robot.parsing.model.statements.LoopControl method*), 383
get_token() (*robot.parsing.model.statements.Metadata method*), 354
get_token() (*robot.parsing.model.statements.MultiValue method*), 348
get_token() (*robot.parsing.model.statements.NoArgumentHeader method*), 377
get_token() (*robot.parsing.model.statements.ResourceImport method*), 352
get_token() (*robot.parsing.model.statements.Return method*), 370
get_token() (*robot.parsing.model.statements.ReturnStatement method*), 382
get_token() (*robot.parsing.model.statements.SectionHeader method*), 350
get_token() (*robot.parsing.model.statements.Setup method*), 365
get_token() (*robot.parsing.model.statements.SingleValue method*), 347
get_token() (*robot.parsing.model.statements.Statement*

method), 346
get_token() (*robot.parsing.model.statements.SuiteSetup method*), 357
get_token() (*robot.parsing.model.statements.SuiteTeardown method*), 358
get_token() (*robot.parsing.model.statements.Tags method*), 366
get_token() (*robot.parsing.model.statements.Teardown method*), 365
get_token() (*robot.parsing.model.statements.Template method*), 367
get_token() (*robot.parsing.model.statements.TemplateArguments method*), 371
get_token() (*robot.parsing.model.statements.TestCaseName method*), 363
get_token() (*robot.parsing.model.statements.TestSetup method*), 359
get_token() (*robot.parsing.model.statements.TestTeardown method*), 360
get_token() (*robot.parsing.model.statements.TestTemplate method*), 361
get_token() (*robot.parsing.model.statements.TestTimeout method*), 361
get_token() (*robot.parsing.model.statements.Timeout method*), 368
get_token() (*robot.parsing.model.statements.TryHeader method*), 378
get_token() (*robot.parsing.model.statements.Variable method*), 362
get_token() (*robot.parsing.model.statements.VariablesImport method*), 352
get_token() (*robot.parsing.model.statements.WhileHeader method*), 381
get_tokens() (*in module robot.parsing.lexer.lexer*), 325
get_tokens() (*robot.parsing.lexer.lexer.Lexer method*), 326
get_tokens() (*robot.parsing.model.statements.Arguments method*), 369
get_tokens() (*robot.parsing.model.statements.Break method*), 385
get_tokens() (*robot.parsing.model.statements.Comment method*), 385
get_tokens() (*robot.parsing.model.statements.Config method*), 386
get_tokens() (*robot.parsing.model.statements.Continue method*), 384
get_tokens() (*robot.parsing.model.statements.DefaultTags method*), 356
get_tokens() (*robot.parsing.model.statements.Documentation method*), 353
get_tokens() (*robot.parsing.model.statements.DocumentationOrMetadata method*), 347
get_tokens() (*robot.parsing.model.statements.ElseHeader method*), 376
get_tokens() (*robot.parsing.model.statements.ElseIfHeader method*), 376
get_tokens() (*robot.parsing.model.statements.EmptyLine method*), 388
get_tokens() (*robot.parsing.model.statements.End method*), 381
get_tokens() (*robot.parsing.model.statements.Error method*), 387
get_tokens() (*robot.parsing.model.statements.ExceptHeader method*), 379
get_tokens() (*robot.parsing.model.statements.FinallyHeader method*), 380
get_tokens() (*robot.parsing.model.statements.Fixture method*), 349
get_tokens() (*robot.parsing.model.statements.ForceTags method*), 355
get_tokens() (*robot.parsing.model.statements.ForHeader method*), 372
get_tokens() (*robot.parsing.model.statements.IfElseHeader method*), 373
get_tokens() (*robot.parsing.model.statements.IfHeader method*), 374
get_tokens() (*robot.parsing.model.statements.InlineIfHeader method*), 375
get_tokens() (*robot.parsing.model.statements.KeywordCall method*), 371
get_tokens() (*robot.parsing.model.statements.KeywordName method*), 364
get_tokens() (*robot.parsing.model.statements.KeywordTags method*), 357
get_tokens() (*robot.parsing.model.statements.LibraryImport method*), 351
get_tokens() (*robot.parsing.model.statements.LoopControl method*), 383
get_tokens() (*robot.parsing.model.statements.Metadata method*), 354
get_tokens() (*robot.parsing.model.statements.MultiValue method*), 348
get_tokens() (*robot.parsing.model.statements.NoArgumentHeader method*), 377
get_tokens() (*robot.parsing.model.statements.ResourceImport method*), 352
get_tokens() (*robot.parsing.model.statements.Return method*), 370
get_tokens() (*robot.parsing.model.statements.ReturnStatement method*), 382
get_tokens() (*robot.parsing.model.statements.SectionHeader method*), 350
get_tokens() (*robot.parsing.model.statements.Setup method*), 365
get_tokens() (*robot.parsing.model.statements.SingleValue method*), 347
get_tokens() (*robot.parsing.model.statements.Statement*

method), 346
get_tokens() (*robot.parsing.model.statements.SuiteSetup method*), 357
get_tokens() (*robot.parsing.model.statements.SuiteTeardown method*), 358
get_tokens() (*robot.parsing.model.statements.Tags method*), 366
get_tokens() (*robot.parsing.model.statements.Teardown method*), 366
get_tokens() (*robot.parsing.model.statements.Template method*), 367
get_tokens() (*robot.parsing.model.statements.TemplateArguments method*), 371
get_tokens() (*robot.parsing.model.statements.TestCaseName method*), 363
get_tokens() (*robot.parsing.model.statements.TestSetup method*), 359
get_tokens() (*robot.parsing.model.statements.TestTeardown method*), 360
get_tokens() (*robot.parsing.model.statements.TestTemplate method*), 361
get_tokens() (*robot.parsing.model.statements.TestTimeout method*), 362
get_tokens() (*robot.parsing.model.statements.Timeout method*), 368
get_tokens() (*robot.parsing.model.statements.TryHeader method*), 378
get_tokens() (*robot.parsing.model.statements.Variable method*), 362
get_tokens() (*robot.parsing.model.statements.VariablesImport method*), 352
get_tokens() (*robot.parsing.model.statements.WhileHeader method*), 381
get_value() (*robot.parsing.model.statements.Arguments method*), 369
get_value() (*robot.parsing.model.statements.Break method*), 385
get_value() (*robot.parsing.model.statements.Comment method*), 386
get_value() (*robot.parsing.model.statements.Config method*), 386
get_value() (*robot.parsing.model.statements.Continue method*), 384
get_value() (*robot.parsing.model.statements.DefaultTags method*), 356
get_value() (*robot.parsing.model.statements.Documentation method*), 353
get_value() (*robot.parsing.model.statements.DocumentationOrMetadata method*), 347
get_value() (*robot.parsing.model.statements.ElseHeader method*), 376
get_value() (*robot.parsing.model.statements.ElseIfHeader method*), 376
get_value() (*robot.parsing.model.statements.EmptyLine method*), 388
get_value() (*robot.parsing.model.statements.End method*), 381
get_value() (*robot.parsing.model.statements.Error method*), 387
get_value() (*robot.parsing.model.statements.ExceptHeader method*), 379
get_value() (*robot.parsing.model.statements.FinallyHeader method*), 380
get_value() (*robot.parsing.model.statements.Fixture method*), 349
get_value() (*robot.parsing.model.statements.ForceTags method*), 355
get_value() (*robot.parsing.model.statements.ForHeader method*), 372
get_value() (*robot.parsing.model.statements.IfElseHeader method*), 373
get_value() (*robot.parsing.model.statements.IfHeader method*), 374
get_value() (*robot.parsing.model.statements.InlineIfHeader method*), 375
get_value() (*robot.parsing.model.statements.KeywordCall method*), 371
get_value() (*robot.parsing.model.statements.KeywordName method*), 364
get_value() (*robot.parsing.model.statements.KeywordTags method*), 357
get_value() (*robot.parsing.model.statements.LibraryImport method*), 351
get_value() (*robot.parsing.model.statements.LoopControl method*), 383
get_value() (*robot.parsing.model.statements.Metadata method*), 354
get_value() (*robot.parsing.model.statements.MultiValue method*), 348
get_value() (*robot.parsing.model.statements.NoArgumentHeader method*), 377
get_value() (*robot.parsing.model.statements.ResourceImport method*), 352
get_value() (*robot.parsing.model.statements.Return method*), 370
get_value() (*robot.parsing.model.statements.ReturnStatement method*), 382
get_value() (*robot.parsing.model.statements.SectionHeader method*), 350
get_value() (*robot.parsing.model.statements.Setup method*), 365
get_value() (*robot.parsing.model.statements.SingleValue method*), 348
get_value() (*robot.parsing.model.statements.Statement method*), 346
get_value() (*robot.parsing.model.statements.SuiteSetup method*), 357
get_value() (*robot.parsing.model.statements.SuiteTeardown

method), 358
get_value() (*robot.parsing.model.statements.Tags method*), 366
get_value() (*robot.parsing.model.statements.Teardown method*), 366
get_value() (*robot.parsing.model.statements.Template method*), 367
get_value() (*robot.parsing.model.statements.TemplateArguments method*), 371
get_value() (*robot.parsing.model.statements.TestCaseName method*), 363
get_value() (*robot.parsing.model.statements.TestSetup method*), 359
get_value() (*robot.parsing.model.statements.TestTeardown method*), 360
get_value() (*robot.parsing.model.statements.TestTemplate method*), 361
get_value() (*robot.parsing.model.statements.TestTimeout method*), 362
get_value() (*robot.parsing.model.statements.Timeout method*), 368
get_value() (*robot.parsing.model.statements.TryHeader method*), 378
get_value() (*robot.parsing.model.statements.Variable method*), 362
get_value() (*robot.parsing.model.statements.VariablesImport method*), 353
get_value() (*robot.parsing.model.statements.WhileHeader method*), 381
get_value_from_user() (*in module robot.libraries.Dialogs*), 111
get_values() (*robot.parsing.model.statements.Arguments method*), 369
get_values() (*robot.parsing.model.statements.Break method*), 385
get_values() (*robot.parsing.model.statements.Comment method*), 386
get_values() (*robot.parsing.model.statements.Config method*), 386
get_values() (*robot.parsing.model.statements.Continue method*), 384
get_values() (*robot.parsing.model.statements.DefaultTags method*), 356
get_values() (*robot.parsing.model.statements.Documentation method*), 353
get_values() (*robot.parsing.model.statements.DocumentationOrMetadata method*), 347
get_values() (*robot.parsing.model.statements.ElseHeader method*), 377
get_values() (*robot.parsing.model.statements.ElseIfHeader method*), 376
get_values() (*robot.parsing.model.statements.EmptyLine method*), 388
get_values() (*robot.parsing.model.statements.End method*), 381
get_values() (*robot.parsing.model.statements.Error method*), 387
get_values() (*robot.parsing.model.statements.ExceptHeader method*), 379
get_values() (*robot.parsing.model.statements.FinallyHeader method*), 380
get_values() (*robot.parsing.model.statements.Fixture method*), 349
get_values() (*robot.parsing.model.statements.ForceTags method*), 355
get_values() (*robot.parsing.model.statements.ForHeader method*), 372
get_values() (*robot.parsing.model.statements.IfElseHeader method*), 373
get_values() (*robot.parsing.model.statements.IfHeader method*), 374
get_values() (*robot.parsing.model.statements.InlineIfHeader method*), 375
get_values() (*robot.parsing.model.statements.KeywordCall method*), 371
get_values() (*robot.parsing.model.statements.KeywordName method*), 364
get_values() (*robot.parsing.model.statements.KeywordTags method*), 357
get_values() (*robot.parsing.model.statements.LibraryImport method*), 351
get_values() (*robot.parsing.model.statements.LoopControl method*), 383
get_values() (*robot.parsing.model.statements.Metadata method*), 354
get_values() (*robot.parsing.model.statements.MultiValue method*), 348
get_values() (*robot.parsing.model.statements.NoArgumentHeader method*), 377
get_values() (*robot.parsing.model.statements.ResourceImport method*), 352
get_values() (*robot.parsing.model.statements.Return method*), 370
get_values() (*robot.parsing.model.statements.ReturnStatement method*), 382
get_values() (*robot.parsing.model.statements.SectionHeader method*), 350
get_values() (*robot.parsing.model.statements.Setup method*), 365
get_values() (*robot.parsing.model.statements.SingleValue method*), 348
get_values() (*robot.parsing.model.statements.Statement method*), 346
get_values() (*robot.parsing.model.statements.SuiteSetup method*), 358
get_values() (*robot.parsing.model.statements.SuiteTeardown method*), 358
get_values() (*robot.parsing.model.statements.Tags*

method), 366
get_values() (*robot.parsing.model.statements.Teardown method*), 366
get_values() (*robot.parsing.model.statements.Template method*), 367
get_values() (*robot.parsing.model.statements.TemplateArguments method*), 371
get_values() (*robot.parsing.model.statements.TestCaseName method*), 363
get_values() (*robot.parsing.model.statements.TestSetup method*), 359
get_values() (*robot.parsing.model.statements.TestTeardown method*), 360
get_values() (*robot.parsing.model.statements.TestTemplate method*), 361
get_values() (*robot.parsing.model.statements.TestTimeout method*), 362
get_values() (*robot.parsing.model.statements.Timeout method*), 368
get_values() (*robot.parsing.model.statements.TryHeader method*), 378
get_values() (*robot.parsing.model.statements.Variable method*), 362
get_values() (*robot.parsing.model.statements.VariablesImport method*), 353
get_values() (*robot.parsing.model.statements.WhileHeader method*), 382
get_variable_value() (*robot.libraries.BuiltIn.BuiltIn method*), 81
get_variables() (*robot.libraries.BuiltIn.BuiltIn method*), 82
get_version() (*in module robot.version*), 625
getboolean() (*robot.libraries.dialogs_py.InputDialog method*), 175
getboolean() (*robot.libraries.dialogs_py.MessageDialog method*), 161
getboolean() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 203
getboolean() (*robot.libraries.dialogs_py.PassFailDialog method*), 217
getboolean() (*robot.libraries.dialogs_py.SelectionDialog method*), 189
getdouble() (*robot.libraries.dialogs_py.InputDialog method*), 175
getdouble() (*robot.libraries.dialogs_py.MessageDialog method*), 161
getdouble() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 203
getdouble() (*robot.libraries.dialogs_py.PassFailDialog method*), 217
getdouble() (*robot.libraries.dialogs_py.SelectionDialog method*), 189
getint() (*robot.libraries.dialogs_py.InputDialog method*), 175
getint() (*robot.libraries.dialogs_py.MessageDialog method*), 161
getint() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 203
getint() (*robot.libraries.dialogs_py.PassFailDialog method*), 217
getint() (*robot.libraries.dialogs_py.SelectionDialog method*), 189
GetKeywordArguments (*class in robot.running.dynamicmethods*), 551
GetKeywordDocumentation (*class in robot.running.dynamicmethods*), 551
GetKeywordNames (*class in robot.running.dynamicmethods*), 550
GetKeywordSource (*class in robot.running.dynamicmethods*), 551
GetKeywordTags (*class in robot.running.dynamicmethods*), 551
GetKeywordTypes (*class in robot.running.dynamicmethods*), 551
getparser() (*robot.libraries.Remote.TimeoutHTTPSTransport method*), 129
getparser() (*robot.libraries.Remote.TimeoutHTTPTransport method*), 129
getter() (*robot.utils.misc.classproperty method*), 604
getvar() (*robot.libraries.dialogs_py.InputDialog method*), 175
getvar() (*robot.libraries.dialogs_py.MessageDialog method*), 161
getvar() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 203
getvar() (*robot.libraries.dialogs_py.PassFailDialog method*), 217
getvar() (*robot.libraries.dialogs_py.SelectionDialog method*), 189
given_prefixes (*robot.conf.languages.Bg attribute*), 60
given_prefixes (*robot.conf.languages.Bs attribute*), 39
given_prefixes (*robot.conf.languages.Cs attribute*), 36
given_prefixes (*robot.conf.languages.De attribute*), 43
given_prefixes (*robot.conf.languages.En attribute*), 35
given_prefixes (*robot.conf.languages.Es attribute*), 51
given_prefixes (*robot.conf.languages.Fi attribute*), 40
given_prefixes (*robot.conf.languages.Fr attribute*), 42
given_prefixes (*robot.conf.languages.Hi attribute*), 64

given_prefixes (*robot.conf.languages.It attribute*), 63
given_prefixes (*robot.conf.languages.Language attribute*), 33
given_prefixes (*robot.conf.languages.Nl attribute*), 37
given_prefixes (*robot.conf.languages.Pl attribute*), 49
given_prefixes (*robot.conf.languages.Pt attribute*), 46
given_prefixes (*robot.conf.languages.PtBr attribute*), 44
given_prefixes (*robot.conf.languages.Ro attribute*), 61
given_prefixes (*robot.conf.languages.Ru attribute*), 53
given_prefixes (*robot.conf.languages.Sv attribute*), 58
given_prefixes (*robot.conf.languages.Th attribute*), 47
given_prefixes (*robot.conf.languages.Tr attribute*), 57
given_prefixes (*robot.conf.languages.Uk attribute*), 50
given_prefixes (*robot.conf.languages.ZhCn attribute*), 54
given_prefixes (*robot.conf.languages.ZhTw attribute*), 56
glob_escape() (*in module robot.utils.escaping*), 598
GlobalScope (*class in robot.running.libraryscopes*), 553
GlobalVariables (*class in robot.variables.scopes*), 610
GlobalVariableValue (*class in robot.variables.resolvable*), 609
grab_current() (*robot.libraries.dialogs_py.InputDialog method*), 175
grab_current() (*robot.libraries.dialogs_py.MessageDialog method*), 161
grab_current() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 203
grab_current() (*robot.libraries.dialogs_py.PassFailDialog method*), 217
grab_current() (*robot.libraries.dialogs_py.SelectionDialog method*), 189
grab_release() (*robot.libraries.dialogs_py.InputDialog method*), 175
grab_release() (*robot.libraries.dialogs_py.MessageDialog method*), 161
grab_release() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 203
grab_release() (*robot.libraries.dialogs_py.PassFailDialog method*), 217
grab_release() (*robot.libraries.dialogs_py.SelectionDialog method*), 189
grab_set() (*robot.libraries.dialogs_py.InputDialog method*), 175
grab_set() (*robot.libraries.dialogs_py.MessageDialog method*), 161
grab_set() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 203
grab_set() (*robot.libraries.dialogs_py.PassFailDialog method*), 217
grab_set() (*robot.libraries.dialogs_py.SelectionDialog method*), 189
grab_set_global() (*robot.libraries.dialogs_py.InputDialog method*), 175
grab_set_global() (*robot.libraries.dialogs_py.MessageDialog method*), 161
grab_set_global() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 203
grab_set_global() (*robot.libraries.dialogs_py.PassFailDialog method*), 217
grab_set_global() (*robot.libraries.dialogs_py.SelectionDialog method*), 189
grab_status() (*robot.libraries.dialogs_py.InputDialog method*), 175
grab_status() (*robot.libraries.dialogs_py.MessageDialog method*), 161
grab_status() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 203
grab_status() (*robot.libraries.dialogs_py.PassFailDialog method*), 217
grab_status() (*robot.libraries.dialogs_py.SelectionDialog method*), 189
green() (*robot.output.console.highlighting.AnsiHighlighter method*), 303
green() (*robot.output.console.highlighting.DosHighlighter method*), 303
green() (*robot.output.console.highlighting.NoHighlighting method*), 303
grep_file() (*robot.libraries.OperatingSystem.OperatingSystem method*), 115
grid() (*robot.libraries.dialogs_py.InputDialog method*), 175
grid() (*robot.libraries.dialogs_py.MessageDialog method*), 161
grid() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 203
grid() (*robot.libraries.dialogs_py.PassFailDialog method*), 217
grid() (*robot.libraries.dialogs_py.SelectionDialog method*), 189

grid_anchor() (*robot.libraries.dialogs_py.InputDialog method*), 176
grid_anchor() (*robot.libraries.dialogs_py.MessageDialog method*), 162
grid_anchor() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 204
grid_anchor() (*robot.libraries.dialogs_py.PassFailDialog method*), 218
grid_anchor() (*robot.libraries.dialogs_py.SelectionDialog method*), 190
grid_bbox() (*robot.libraries.dialogs_py.InputDialog method*), 176
grid_bbox() (*robot.libraries.dialogs_py.MessageDialog method*), 162
grid_bbox() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 204
grid_bbox() (*robot.libraries.dialogs_py.PassFailDialog method*), 218
grid_bbox() (*robot.libraries.dialogs_py.SelectionDialog method*), 190
grid_columnconfigure() (*robot.libraries.dialogs_py.InputDialog method*), 176
grid_columnconfigure() (*robot.libraries.dialogs_py.MessageDialog method*), 162
grid_columnconfigure() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 204
grid_columnconfigure() (*robot.libraries.dialogs_py.PassFailDialog method*), 218
grid_columnconfigure() (*robot.libraries.dialogs_py.SelectionDialog method*), 190
grid_location() (*robot.libraries.dialogs_py.InputDialog method*), 176
grid_location() (*robot.libraries.dialogs_py.MessageDialog method*), 162
grid_location() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 204
grid_location() (*robot.libraries.dialogs_py.PassFailDialog method*), 218
grid_location() (*robot.libraries.dialogs_py.SelectionDialog method*), 190
grid_propagate() (*robot.libraries.dialogs_py.InputDialog method*), 176
grid_propagate() (*robot.libraries.dialogs_py.MessageDialog method*), 162
grid_propagate() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 204
grid_propagate() (*robot.libraries.dialogs_py.PassFailDialog method*), 218
grid_propagate() (*robot.libraries.dialogs_py.SelectionDialog method*), 190
grid_rowconfigure() (*robot.libraries.dialogs_py.InputDialog method*), 176
grid_rowconfigure() (*robot.libraries.dialogs_py.MessageDialog method*), 162
grid_rowconfigure() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 204
grid_rowconfigure() (*robot.libraries.dialogs_py.PassFailDialog method*), 218
grid_rowconfigure() (*robot.libraries.dialogs_py.SelectionDialog method*), 190
grid_size() (*robot.libraries.dialogs_py.InputDialog method*), 176
grid_size() (*robot.libraries.dialogs_py.MessageDialog method*), 162
grid_size() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 204
grid_size() (*robot.libraries.dialogs_py.PassFailDialog method*), 218
grid_size() (*robot.libraries.dialogs_py.SelectionDialog method*), 190
grid_slaves() (*robot.libraries.dialogs_py.InputDialog method*), 176
grid_slaves() (*robot.libraries.dialogs_py.MessageDialog method*), 162
grid_slaves() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 204
grid_slaves() (*robot.libraries.dialogs_py.PassFailDialog method*), 218
grid_slaves() (*robot.libraries.dialogs_py.SelectionDialog method*), 190
group() (*robot.libraries.dialogs_py.InputDialog method*), 176
group() (*robot.libraries.dialogs_py.MessageDialog method*), 162
group() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 204
group() (*robot.libraries.dialogs_py.PassFailDialog method*), 218
group() (*robot.libraries.dialogs_py.SelectionDialog method*), 190

H

handle() (*robot.output.pyloggingconf.RobotHandler method*), 312
handle() (*robot.running.arguments.argumentresolver.DictToKwargs method*), 529
handle_imports() (*robot.running.namespace.Namespace method*), 577

handle_suite_teardown_failures() (*robot.result.executionresult.CombinedResult method*), 414
handle_suite_teardown_failures() (*robot.result.executionresult.Result method*), 414
handle_suite_teardown_failures() (*robot.result.model.TestSuite method*), 499
handleError() (*robot.output.pyloggingconf.RobotHandler method*), 312
Handler() (*in module robot.running.handlers*), 551
HandlerExecutionFailed, 615
handlers_for() (*robot.running.userkeyword.UserLibrary method*), 589
HandlerStore (*class in robot.running.handlerstore*), 551
handles() (*robot.htmldata.htmlfilewriter.CssFileWriter method*), 67
handles() (*robot.htmldata.htmlfilewriter.GeneratorWriter method*), 67
handles() (*robot.htmldata.htmlfilewriter.JsFileWriter method*), 67
handles() (*robot.htmldata.htmlfilewriter.LineWriter method*), 67
handles() (*robot.htmldata.htmlfilewriter.ModelWriter method*), 67
handles() (*robot.htmldata.jsonwriter.DictDumper method*), 68
handles() (*robot.htmldata.jsonwriter.IntegerDumper method*), 68
handles() (*robot.htmldata.jsonwriter.MappingDumper method*), 68
handles() (*robot.htmldata.jsonwriter.NoneDumper method*), 68
handles() (*robot.htmldata.jsonwriter.StringDumper method*), 68
handles() (*robot.htmldata.jsonwriter.TupleListDumper method*), 68
handles() (*robot.libdocpkg.consoleviewer.ConsoleViewer class method*), 69
handles() (*robot.libdocpkg.htmlwriter.LibdocModelWriter method*), 71
handles() (*robot.parsing.lexer.blocklexers.BlockLexer class method*), 318
handles() (*robot.parsing.lexer.blocklexers.CommentSectionLexer class method*), 320
handles() (*robot.parsing.lexer.blocklexers.ErrorSectionLexer class method*), 321
handles() (*robot.parsing.lexer.blocklexers.FileLexer class method*), 319
handles() (*robot.parsing.lexer.blocklexers.ForLexer class method*), 322
handles() (*robot.parsing.lexer.blocklexers.IfLexer class method*), 322
handles() (*robot.parsing.lexer.blocklexers.ImplicitCommentSectionLexer class method*), 320
handles() (*robot.parsing.lexer.blocklexers.InlineIfLexer class method*), 323
handles() (*robot.parsing.lexer.blocklexers.KeywordLexer class method*), 322
handles() (*robot.parsing.lexer.blocklexers.KeywordSectionLexer class method*), 320
handles() (*robot.parsing.lexer.blocklexers.NestedBlockLexer class method*), 322
handles() (*robot.parsing.lexer.blocklexers.SectionLexer class method*), 319
handles() (*robot.parsing.lexer.blocklexers.SettingSectionLexer class method*), 319
handles() (*robot.parsing.lexer.blocklexers.TaskSectionLexer class method*), 320
handles() (*robot.parsing.lexer.blocklexers.TestCaseLexer class method*), 321
handles() (*robot.parsing.lexer.blocklexers.TestCaseSectionLexer class method*), 320
handles() (*robot.parsing.lexer.blocklexers.TestOrKeywordLexer class method*), 321
handles() (*robot.parsing.lexer.blocklexers.TryLexer class method*), 323
handles() (*robot.parsing.lexer.blocklexers.VariableSectionLexer class method*), 319
handles() (*robot.parsing.lexer.blocklexers.WhileLexer class method*), 322
handles() (*robot.parsing.lexer.statementlexers.BreakLexer class method*), 333
handles() (*robot.parsing.lexer.statementlexers.CommentLexer class method*), 330
handles() (*robot.parsing.lexer.statementlexers.CommentSectionHeaderL class method*), 329
handles() (*robot.parsing.lexer.statementlexers.ContinueLexer class method*), 333
handles() (*robot.parsing.lexer.statementlexers.ElseHeaderLexer class method*), 331
handles() (*robot.parsing.lexer.statementlexers.ElseIfHeaderLexer class method*), 331
handles() (*robot.parsing.lexer.statementlexers.EndLexer class method*), 332
handles() (*robot.parsing.lexer.statementlexers.ErrorSectionHeaderLexe class method*), 329
handles() (*robot.parsing.lexer.statementlexers.ExceptHeaderLexer class method*), 332
handles() (*robot.parsing.lexer.statementlexers.FinallyHeaderLexer class method*), 332
handles() (*robot.parsing.lexer.statementlexers.ForHeaderLexer class method*), 331
handles() (*robot.parsing.lexer.statementlexers.IfHeaderLexer class method*), 331
handles() (*robot.parsing.lexer.statementlexers.ImplicitCommentLexer class method*), 330

handles() (*robot.parsing.lexer.statementlexers.InlineIfHeaderLexer class method*), 331
handles() (*robot.parsing.lexer.statementlexers.KeywordCallLexer class method*), 330
handles() (*robot.parsing.lexer.statementlexers.KeywordSectionHeaderLexer class method*), 329
handles() (*robot.parsing.lexer.statementlexers.Lexer class method*), 327
handles() (*robot.parsing.lexer.statementlexers.ReturnLexer class method*), 332
handles() (*robot.parsing.lexer.statementlexers.SectionHeaderLexer class method*), 328
handles() (*robot.parsing.lexer.statementlexers.SettingLexer class method*), 330
handles() (*robot.parsing.lexer.statementlexers.SettingSectionHeaderLexer class method*), 328
handles() (*robot.parsing.lexer.statementlexers.SingleType class method*), 328
handles() (*robot.parsing.lexer.statementlexers.StatementLexer class method*), 328
handles() (*robot.parsing.lexer.statementlexers.TaskSectionHeaderLexer class method*), 329
handles() (*robot.parsing.lexer.statementlexers.TestCaseSectionHeaderLexer class method*), 329
handles() (*robot.parsing.lexer.statementlexers.TestOrKeywordSettingLexer class method*), 330
handles() (*robot.parsing.lexer.statementlexers.TryHeaderLexer class method*), 332
handles() (*robot.parsing.lexer.statementlexers.TypeAndArguments class method*), 328
handles() (*robot.parsing.lexer.statementlexers.VariableLexer class method*), 330
handles() (*robot.parsing.lexer.statementlexers.VariableSectionHeaderLexer class method*), 328
handles() (*robot.parsing.lexer.statementlexers.WhileHeaderLexer class method*), 332
handles() (*robot.parsing.parser.blockparsers.BlockParser method*), 389
handles() (*robot.parsing.parser.blockparsers.ForParser method*), 390
handles() (*robot.parsing.parser.blockparsers.IfParser method*), 390
handles() (*robot.parsing.parser.blockparsers.KeywordParser method*), 389
handles() (*robot.parsing.parser.blockparsers.NestedBlockParser method*), 390
handles() (*robot.parsing.parser.blockparsers.Parser method*), 389
handles() (*robot.parsing.parser.blockparsers.TestCaseParser method*), 389
handles() (*robot.parsing.parser.blockparsers.TryParser method*), 390
handles() (*robot.parsing.parser.blockparsers.WhileParser method*), 390
handles() (*robot.parsing.parser.fileparser.CommentSectionParser method*), 391
handles() (*robot.parsing.parser.fileparser.FileParser method*), 390
handles() (*robot.parsing.parser.fileparser.ImplicitCommentSectionParser method*), 391
handles() (*robot.parsing.parser.fileparser.KeywordSectionParser method*), 391
handles() (*robot.parsing.parser.fileparser.SectionParser method*), 390
handles() (*robot.parsing.parser.fileparser.SettingSectionParser method*), 391
handles() (*robot.parsing.parser.fileparser.TestCaseSectionParser method*), 391
handles() (*robot.parsing.parser.fileparser.VariableSectionParser method*), 391
handles() (*robot.reporting.logreportwriters.RobotModelWriter method*), 395
handles() (*robot.running.arguments.typeconverters.BooleanConverter class method*), 533
handles() (*robot.running.arguments.typeconverters.ByteArrayConverter class method*), 535
handles() (*robot.running.arguments.typeconverters.BytesConverter class method*), 535
handles() (*robot.running.arguments.typeconverters.CombinedConverter class method*), 540
handles() (*robot.running.arguments.typeconverters.CustomConverter class method*), 541
handles() (*robot.running.arguments.typeconverters.DateConverter class method*), 536
handles() (*robot.running.arguments.typeconverters.DateTimeConverter class method*), 536
handles() (*robot.running.arguments.typeconverters.DecimalConverter class method*), 534
handles() (*robot.running.arguments.typeconverters.DictionaryConverter class method*), 539
handles() (*robot.running.arguments.typeconverters.EnumConverter class method*), 532
handles() (*robot.running.arguments.typeconverters.FloatConverter class method*), 534
handles() (*robot.running.arguments.typeconverters.FrozenSetConverter class method*), 540
handles() (*robot.running.arguments.typeconverters.IntegerConverter class method*), 533
handles() (*robot.running.arguments.typeconverters.ListConverter class method*), 538
handles() (*robot.running.arguments.typeconverters.NoneConverter class method*), 537
handles() (*robot.running.arguments.typeconverters.PathConverter class method*), 537
handles() (*robot.running.arguments.typeconverters.SetConverter class method*), 539
handles() (*robot.running.arguments.typeconverters.StringConverter class method*), 533

handles() (*robot.running.arguments.typeconverters.TimedeltaConverter class method*), 536
handles() (*robot.running.arguments.typeconverters.TupleConverter class method*), 538
handles() (*robot.running.arguments.typeconverters.TypeConverter class method*), 532
handles() (*robot.running.arguments.typeconverters.TypedDictConverter class method*), 538
handles() (*robot.testdoc.TestdocModelWriter method*), 624
handles() (*robot.utils.htmlformatters.HeaderFormatter method*), 599
handles() (*robot.utils.htmlformatters.LineFormatter method*), 599
handles() (*robot.utils.htmlformatters.ListFormatter method*), 600
handles() (*robot.utils.htmlformatters.ParagraphFormatter method*), 600
handles() (*robot.utils.htmlformatters.PreformattedFormatter method*), 600
handles() (*robot.utils.htmlformatters.RulerFormatter method*), 599
handles() (*robot.utils.htmlformatters.TableFormatter method*), 600
handles() (*robot.utils.importer.ByPathImporter method*), 601
handles() (*robot.utils.importer.DottedImporter method*), 602
handles() (*robot.utils.importer.NonDottedImporter method*), 602
handles_types (*robot.parsing.model.statements.Arguments attribute*), 369
handles_types (*robot.parsing.model.statements.Break attribute*), 385
handles_types (*robot.parsing.model.statements.Comment attribute*), 386
handles_types (*robot.parsing.model.statements.Config attribute*), 386
handles_types (*robot.parsing.model.statements.Continue attribute*), 384
handles_types (*robot.parsing.model.statements.DefaultTags attribute*), 356
handles_types (*robot.parsing.model.statements.Documentation attribute*), 353
handles_types (*robot.parsing.model.statements.DocumentationOrMetadata attribute*), 347
handles_types (*robot.parsing.model.statements.ElseHeader attribute*), 377
handles_types (*robot.parsing.model.statements.ElseIfHeader attribute*), 376
handles_types (*robot.parsing.model.statements.EmptyLine attribute*), 388
handles_types (*robot.parsing.model.statements.End attribute*), 381
handles_types (*robot.parsing.model.statements.Error attribute*), 387
handles_types (*robot.parsing.model.statements.ExceptHeader attribute*), 379
handles_types (*robot.parsing.model.statements.FinallyHeader attribute*), 380
handles_types (*robot.parsing.model.statements.Fixture attribute*), 349
handles_types (*robot.parsing.model.statements.ForceTags attribute*), 355
handles_types (*robot.parsing.model.statements.ForHeader attribute*), 372
handles_types (*robot.parsing.model.statements.IfElseHeader attribute*), 373
handles_types (*robot.parsing.model.statements.IfHeader attribute*), 374
handles_types (*robot.parsing.model.statements.InlineIfHeader attribute*), 375
handles_types (*robot.parsing.model.statements.KeywordCall attribute*), 371
handles_types (*robot.parsing.model.statements.KeywordName attribute*), 364
handles_types (*robot.parsing.model.statements.KeywordTags attribute*), 357
handles_types (*robot.parsing.model.statements.LibraryImport attribute*), 351
handles_types (*robot.parsing.model.statements.LoopControl attribute*), 383
handles_types (*robot.parsing.model.statements.Metadata attribute*), 354
handles_types (*robot.parsing.model.statements.MultiValue attribute*), 348
handles_types (*robot.parsing.model.statements.NoArgumentHeader attribute*), 377
handles_types (*robot.parsing.model.statements.ResourceImport attribute*), 352
handles_types (*robot.parsing.model.statements.Return attribute*), 370
handles_types (*robot.parsing.model.statements.ReturnStatement attribute*), 382
handles_types (*robot.parsing.model.statements.SectionHeader attribute*), 350
handles_types (*robot.parsing.model.statements.Setup attribute*), 365
handles_types (*robot.parsing.model.statements.SingleValue attribute*), 348
handles_types (*robot.parsing.model.statements.Statement attribute*), 346
handles_types (*robot.parsing.model.statements.SuiteSetup attribute*), 358
handles_types (*robot.parsing.model.statements.SuiteTeardown attribute*), 358
handles_types (*robot.parsing.model.statements.Tags attribute*), 367

handles_types (*robot.parsing.model.statements.Teardown attribute*), 366
handles_types (*robot.parsing.model.statements.Template attribute*), 367
handles_types (*robot.parsing.model.statements.TemplateArguments attribute*), 371
handles_types (*robot.parsing.model.statements.TestCaseName attribute*), 363
handles_types (*robot.parsing.model.statements.TestSetup attribute*), 359
handles_types (*robot.parsing.model.statements.TestTeardown attribute*), 360
handles_types (*robot.parsing.model.statements.TestTemplate attribute*), 361
handles_types (*robot.parsing.model.statements.TestTimeout attribute*), 362
handles_types (*robot.parsing.model.statements.Timeout attribute*), 368
handles_types (*robot.parsing.model.statements.TryHeader attribute*), 378
handles_types (*robot.parsing.model.statements.Variable attribute*), 362
handles_types (*robot.parsing.model.statements.VariablesImport attribute*), 353
handles_types (*robot.parsing.model.statements.WhileHeader attribute*), 382
has_setup (*robot.model.body.BodyItem attribute*), 228
has_setup (*robot.model.control.Break attribute*), 250
has_setup (*robot.model.control.Continue attribute*), 249
has_setup (*robot.model.control.For attribute*), 240
has_setup (*robot.model.control.If attribute*), 243
has_setup (*robot.model.control.IfBranch attribute*), 242
has_setup (*robot.model.control.Return attribute*), 247
has_setup (*robot.model.control.Try attribute*), 246
has_setup (*robot.model.control.TryBranch attribute*), 245
has_setup (*robot.model.control.While attribute*), 241
has_setup (*robot.model.keyword.Keyword attribute*), 261
has_setup (*robot.model.message.Message attribute*), 264
has_setup (*robot.model.testcase.TestCase attribute*), 283
has_setup (*robot.model.testsuite.TestSuite attribute*), 286
has_setup (*robot.output.loggerhelper.Message attribute*), 310
has_setup (*robot.result.model.Break attribute*), 489
has_setup (*robot.result.model.Continue attribute*), 487
has_setup (*robot.result.model.For attribute*), 468
has_setup (*robot.result.model.ForIteration attribute*), 466
has_setup (*robot.result.model.If attribute*), 477
has_setup (*robot.result.model.IfBranch attribute*), 475
has_setup (*robot.result.model.Keyword attribute*), 492
has_setup (*robot.result.model.Message attribute*), 464
has_setup (*robot.result.model.Return attribute*), 484
has_setup (*robot.result.model.TestCase attribute*), 494
has_setup (*robot.result.model.TestSuite attribute*), 497
has_setup (*robot.result.model.Try attribute*), 482
has_setup (*robot.result.model.TryBranch attribute*), 480
has_setup (*robot.result.model.While attribute*), 473
has_setup (*robot.result.model.WhileIteration attribute*), 471
has_setup (*robot.running.model.Break attribute*), 570
has_setup (*robot.running.model.Continue attribute*), 568
has_setup (*robot.running.model.For attribute*), 558
has_setup (*robot.running.model.If attribute*), 563
has_setup (*robot.running.model.IfBranch attribute*), 561
has_setup (*robot.running.model.Keyword attribute*), 556
has_setup (*robot.running.model.Return attribute*), 567
has_setup (*robot.running.model.TestCase attribute*), 571
has_setup (*robot.running.model.TestSuite attribute*), 574
has_setup (*robot.running.model.Try attribute*), 566
has_setup (*robot.running.model.TryBranch attribute*), 564
has_setup (*robot.running.model.While attribute*), 560
has_teardown (*robot.model.body.BodyItem attribute*), 228
has_teardown (*robot.model.control.Break attribute*), 250
has_teardown (*robot.model.control.Continue attribute*), 249
has_teardown (*robot.model.control.For attribute*), 240
has_teardown (*robot.model.control.If attribute*), 244
has_teardown (*robot.model.control.IfBranch attribute*), 242
has_teardown (*robot.model.control.Return attribute*), 247
has_teardown (*robot.model.control.Try attribute*), 246
has_teardown (*robot.model.control.TryBranch attribute*), 245
has_teardown (*robot.model.control.While attribute*), 241
has_teardown (*robot.model.keyword.Keyword attribute*), 260

has_teardown (*robot.model.message.Message attribute*), 264
has_teardown (*robot.model.testcase.TestCase attribute*), 283
has_teardown (*robot.model.testsuite.TestSuite attribute*), 286
has_teardown (*robot.output.loggerhelper.Message attribute*), 310
has_teardown (*robot.result.model.Break attribute*), 489
has_teardown (*robot.result.model.Continue attribute*), 487
has_teardown (*robot.result.model.For attribute*), 468
has_teardown (*robot.result.model.ForIteration attribute*), 466
has_teardown (*robot.result.model.If attribute*), 477
has_teardown (*robot.result.model.IfBranch attribute*), 475
has_teardown (*robot.result.model.Keyword attribute*), 492
has_teardown (*robot.result.model.Message attribute*), 464
has_teardown (*robot.result.model.Return attribute*), 484
has_teardown (*robot.result.model.TestCase attribute*), 494
has_teardown (*robot.result.model.TestSuite attribute*), 497
has_teardown (*robot.result.model.Try attribute*), 482
has_teardown (*robot.result.model.TryBranch attribute*), 480
has_teardown (*robot.result.model.While attribute*), 473
has_teardown (*robot.result.model.WhileIteration attribute*), 471
has_teardown (*robot.running.model.Break attribute*), 570
has_teardown (*robot.running.model.Continue attribute*), 568
has_teardown (*robot.running.model.For attribute*), 558
has_teardown (*robot.running.model.If attribute*), 563
has_teardown (*robot.running.model.IfBranch attribute*), 561
has_teardown (*robot.running.model.Keyword attribute*), 556
has_teardown (*robot.running.model.Return attribute*), 567
has_teardown (*robot.running.model.TestCase attribute*), 571
has_teardown (*robot.running.model.TestSuite attribute*), 574
has_teardown (*robot.running.model.Try attribute*), 566
has_teardown (*robot.running.model.TryBranch attribute*), 564
has_teardown (*robot.running.model.While attribute*), 560
has_tests (*robot.model.testsuite.TestSuite attribute*), 286
has_tests (*robot.result.model.TestSuite attribute*), 497
has_tests (*robot.running.model.TestSuite attribute*), 574
HEADER_TOKENS (*robot.parsing.lexer.tokens.END attribute*), 339
HEADER_TOKENS (*robot.parsing.lexer.tokens.EOS attribute*), 336
HEADER_TOKENS (*robot.parsing.lexer.tokens.Token attribute*), 335
HeaderAndBody (*class in robot.parsing.model.blocks*), 340
HeaderFormatter (*class in robot.utils.htmlformatters*), 599
headers (*robot.conf.languages.Bg attribute*), 60
headers (*robot.conf.languages.Bs attribute*), 39
headers (*robot.conf.languages.Cs attribute*), 36
headers (*robot.conf.languages.De attribute*), 43
headers (*robot.conf.languages.En attribute*), 35
headers (*robot.conf.languages.Es attribute*), 52
headers (*robot.conf.languages.Fi attribute*), 41
headers (*robot.conf.languages.Fr attribute*), 42
headers (*robot.conf.languages.Hi attribute*), 64
headers (*robot.conf.languages.It attribute*), 63
headers (*robot.conf.languages.Language attribute*), 34
headers (*robot.conf.languages.Nl attribute*), 38
headers (*robot.conf.languages.Pl attribute*), 49
headers (*robot.conf.languages.Pt attribute*), 46
headers (*robot.conf.languages.PtBr attribute*), 45
headers (*robot.conf.languages.Ro attribute*), 62
headers (*robot.conf.languages.Ru attribute*), 53
headers (*robot.conf.languages.Sv attribute*), 59
headers (*robot.conf.languages.Th attribute*), 48
headers (*robot.conf.languages.Tr attribute*), 57
headers (*robot.conf.languages.Uk attribute*), 50
headers (*robot.conf.languages.ZhCn attribute*), 55
headers (*robot.conf.languages.ZhTw attribute*), 56
Hi (*class in robot.conf.languages*), 63
highlight() (*robot.output.console.highlighting.HighlightingStream method*), 303
Highlighter() (*in module robot.output.console.highlighting*), 303
HighlightingStream (*class in robot.output.console.highlighting*), 303
html (*robot.model.message.Message attribute*), 262
html (*robot.output.loggerhelper.Message attribute*), 310
html (*robot.result.model.Message attribute*), 464
html() (*robot.libdocpkg.htmlutils.DocFormatter method*), 70

html() (*robot.reporting.jsbuildingcontext.JsBuildingContext method*), 393
html_chars (*robot.libdocpkg.htmlutils.HtmlToText attribute*), 70
html_escape() (*in module robot.utils.markuputils*), 602
html_format() (*in module robot.utils.markuputils*), 602
html_message (*robot.model.message.Message attribute*), 262
html_message (*robot.output.loggerhelper.Message attribute*), 310
html_message (*robot.result.model.Message attribute*), 464
html_tags (*robot.libdocpkg.htmlutils.HtmlToText attribute*), 70
html_to_plain_text() (*robot.libdocpkg.htmlutils.HtmlToText method*), 71
HtmlFileWriter (*class in robot.htmldata.htmlfilewriter*), 67
HtmlFormatter (*class in robot.utils.htmlformatters*), 599
HtmlTemplate (*class in robot.htmldata.template*), 68
HtmlToText (*class in robot.libdocpkg.htmlutils*), 70
HtmlWriter (*class in robot.utils.markupwriters*), 602

I

iconbitmap() (*robot.libraries.dialogs_py.InputDialog method*), 176
iconbitmap() (*robot.libraries.dialogs_py.MessageDialog method*), 162
iconbitmap() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 204
iconbitmap() (*robot.libraries.dialogs_py.PassFailDialog method*), 218
iconbitmap() (*robot.libraries.dialogs_py.SelectionDialog method*), 190
iconify() (*robot.libraries.dialogs_py.InputDialog method*), 176
iconify() (*robot.libraries.dialogs_py.MessageDialog method*), 162
iconify() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 204
iconify() (*robot.libraries.dialogs_py.PassFailDialog method*), 218
iconify() (*robot.libraries.dialogs_py.SelectionDialog method*), 190
iconmask() (*robot.libraries.dialogs_py.InputDialog method*), 176
iconmask() (*robot.libraries.dialogs_py.MessageDialog method*), 162
iconmask() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 204
iconmask() (*robot.libraries.dialogs_py.PassFailDialog method*), 218
iconmask() (*robot.libraries.dialogs_py.SelectionDialog method*), 190
iconname() (*robot.libraries.dialogs_py.InputDialog method*), 177
iconname() (*robot.libraries.dialogs_py.MessageDialog method*), 163
iconname() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 205
iconname() (*robot.libraries.dialogs_py.PassFailDialog method*), 219
iconname() (*robot.libraries.dialogs_py.SelectionDialog method*), 191
iconphoto() (*robot.libraries.dialogs_py.InputDialog method*), 177
iconphoto() (*robot.libraries.dialogs_py.MessageDialog method*), 163
iconphoto() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 205
iconphoto() (*robot.libraries.dialogs_py.PassFailDialog method*), 219
iconphoto() (*robot.libraries.dialogs_py.SelectionDialog method*), 191
iconposition() (*robot.libraries.dialogs_py.InputDialog method*), 177
iconposition() (*robot.libraries.dialogs_py.MessageDialog method*), 163
iconposition() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 205
iconposition() (*robot.libraries.dialogs_py.PassFailDialog method*), 219
iconposition() (*robot.libraries.dialogs_py.SelectionDialog method*), 191
iconwindow() (*robot.libraries.dialogs_py.InputDialog method*), 177
iconwindow() (*robot.libraries.dialogs_py.MessageDialog method*), 163
iconwindow() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 205
iconwindow() (*robot.libraries.dialogs_py.PassFailDialog method*), 219
iconwindow() (*robot.libraries.dialogs_py.SelectionDialog method*), 191
id (*robot.model.body.BodyItem attribute*), 228
id (*robot.model.control.Break attribute*), 250
id (*robot.model.control.Continue attribute*), 249
id (*robot.model.control.For attribute*), 240
id (*robot.model.control.If attribute*), 243
id (*robot.model.control.IfBranch attribute*), 241
id (*robot.model.control.Return attribute*), 247
id (*robot.model.control.Try attribute*), 245
id (*robot.model.control.TryBranch attribute*), 244
id (*robot.model.control.While attribute*), 241

id (*robot.model.keyword.Keyword attribute*), 261
id (*robot.model.message.Message attribute*), 263
id (*robot.model.stats.SuiteStat attribute*), 275
id (*robot.model.testcase.TestCase attribute*), 284
id (*robot.model.testsuite.TestSuite attribute*), 286
id (*robot.output.loggerhelper.Message attribute*), 310
id (*robot.result.executionerrors.ExecutionErrors attribute*), 412
id (*robot.result.model.Break attribute*), 489
id (*robot.result.model.Continue attribute*), 487
id (*robot.result.model.For attribute*), 468
id (*robot.result.model.ForIteration attribute*), 466
id (*robot.result.model.If attribute*), 478
id (*robot.result.model.IfBranch attribute*), 475
id (*robot.result.model.Keyword attribute*), 492
id (*robot.result.model.Message attribute*), 464
id (*robot.result.model.Return attribute*), 484
id (*robot.result.model.TestCase attribute*), 494
id (*robot.result.model.TestSuite attribute*), 497
id (*robot.result.model.Try attribute*), 482
id (*robot.result.model.TryBranch attribute*), 480
id (*robot.result.model.While attribute*), 473
id (*robot.result.model.WhileIteration attribute*), 471
id (*robot.running.model.Break attribute*), 570
id (*robot.running.model.Continue attribute*), 568
id (*robot.running.model.For attribute*), 559
id (*robot.running.model.If attribute*), 563
id (*robot.running.model.IfBranch attribute*), 561
id (*robot.running.model.Keyword attribute*), 557
id (*robot.running.model.Return attribute*), 567
id (*robot.running.model.TestCase attribute*), 571
id (*robot.running.model.TestSuite attribute*), 574
id (*robot.running.model.Try attribute*), 566
id (*robot.running.model.TryBranch attribute*), 564
id (*robot.running.model.While attribute*), 560
identifiers (*robot.variables.finders.EmptyFinder attribute*), 608
identifiers (*robot.variables.finders.EnvironmentFinder attribute*), 608
identifiers (*robot.variables.finders.ExtendedFinder attribute*), 608
identifiers (*robot.variables.finders.InlinePythonFinder attribute*), 608
identifiers (*robot.variables.finders.NumberFinder attribute*), 608
identifiers (*robot.variables.finders.StoredFinder attribute*), 608
If (*class in robot.model.control*), 242
If (*class in robot.parsing.model.blocks*), 343
If (*class in robot.result.model*), 476
If (*class in robot.running.model*), 562
IF (*robot.model.body.BodyItem attribute*), 228
IF (*robot.model.control.Break attribute*), 249
IF (*robot.model.control.Continue attribute*), 248
IF (*robot.model.control.For attribute*), 239
IF (*robot.model.control.If attribute*), 243
IF (*robot.model.control.IfBranch attribute*), 242
IF (*robot.model.control.Return attribute*), 247
IF (*robot.model.control.Try attribute*), 246
IF (*robot.model.control.TryBranch attribute*), 244
IF (*robot.model.control.While attribute*), 240
IF (*robot.model.keyword.Keyword attribute*), 260
IF (*robot.model.message.Message attribute*), 263
IF (*robot.output.loggerhelper.Message attribute*), 310
IF (*robot.parsing.lexer.tokens.END attribute*), 339
IF (*robot.parsing.lexer.tokens.EOS attribute*), 336
IF (*robot.parsing.lexer.tokens.Token attribute*), 335
IF (*robot.result.model.Break attribute*), 488
IF (*robot.result.model.Continue attribute*), 486
IF (*robot.result.model.For attribute*), 467
IF (*robot.result.model.ForIteration attribute*), 465
IF (*robot.result.model.If attribute*), 476
IF (*robot.result.model.IfBranch attribute*), 474
IF (*robot.result.model.Keyword attribute*), 491
IF (*robot.result.model.Message attribute*), 463
IF (*robot.result.model.Return attribute*), 483
IF (*robot.result.model.Try attribute*), 481
IF (*robot.result.model.TryBranch attribute*), 479
IF (*robot.result.model.While attribute*), 472
IF (*robot.result.model.WhileIteration attribute*), 470
IF (*robot.running.model.Break attribute*), 569
IF (*robot.running.model.Continue attribute*), 568
IF (*robot.running.model.For attribute*), 558
IF (*robot.running.model.If attribute*), 562
IF (*robot.running.model.IfBranch attribute*), 561
IF (*robot.running.model.Keyword attribute*), 556
IF (*robot.running.model.Return attribute*), 566
IF (*robot.running.model.Try attribute*), 565
IF (*robot.running.model.TryBranch attribute*), 563
IF (*robot.running.model.While attribute*), 559
if_class (*robot.model.body.BaseBody attribute*), 229
if_class (*robot.model.body.Body attribute*), 231
if_class (*robot.model.body.Branches attribute*), 233
if_class (*robot.result.model.Body attribute*), 459
if_class (*robot.result.model.Branches attribute*), 460
if_class (*robot.result.model.Iterations attribute*), 462
if_class (*robot.running.model.Body attribute*), 554
IF_ELSE_ROOT (*robot.model.body.BodyItem attribute*), 228
IF_ELSE_ROOT (*robot.model.control.Break attribute*), 249
IF_ELSE_ROOT (*robot.model.control.Continue attribute*), 248
IF_ELSE_ROOT (*robot.model.control.For attribute*), 239
IF_ELSE_ROOT (*robot.model.control.If attribute*), 243
IF_ELSE_ROOT (*robot.model.control.IfBranch attribute*), 242

IF_ELSE_ROOT (*robot.model.control.Return attribute*), 247
IF_ELSE_ROOT (*robot.model.control.Try attribute*), 246
IF_ELSE_ROOT (*robot.model.control.TryBranch attribute*), 244
IF_ELSE_ROOT (*robot.model.control.While attribute*), 240
IF_ELSE_ROOT (*robot.model.keyword.Keyword attribute*), 260
IF_ELSE_ROOT (*robot.model.message.Message attribute*), 263
IF_ELSE_ROOT (*robot.output.loggerhelper.Message attribute*), 310
IF_ELSE_ROOT (*robot.result.model.Break attribute*), 488
IF_ELSE_ROOT (*robot.result.model.Continue attribute*), 486
IF_ELSE_ROOT (*robot.result.model.For attribute*), 467
IF_ELSE_ROOT (*robot.result.model.ForIteration attribute*), 465
IF_ELSE_ROOT (*robot.result.model.If attribute*), 476
IF_ELSE_ROOT (*robot.result.model.IfBranch attribute*), 474
IF_ELSE_ROOT (*robot.result.model.Keyword attribute*), 491
IF_ELSE_ROOT (*robot.result.model.Message attribute*), 463
IF_ELSE_ROOT (*robot.result.model.Return attribute*), 483
IF_ELSE_ROOT (*robot.result.model.Try attribute*), 481
IF_ELSE_ROOT (*robot.result.model.TryBranch attribute*), 479
IF_ELSE_ROOT (*robot.result.model.While attribute*), 472
IF_ELSE_ROOT (*robot.result.model.WhileIteration attribute*), 470
IF_ELSE_ROOT (*robot.running.model.Break attribute*), 569
IF_ELSE_ROOT (*robot.running.model.Continue attribute*), 568
IF_ELSE_ROOT (*robot.running.model.For attribute*), 558
IF_ELSE_ROOT (*robot.running.model.If attribute*), 562
IF_ELSE_ROOT (*robot.running.model.IfBranch attribute*), 561
IF_ELSE_ROOT (*robot.running.model.Keyword attribute*), 556
IF_ELSE_ROOT (*robot.running.model.Return attribute*), 566
IF_ELSE_ROOT (*robot.running.model.Try attribute*), 565
IF_ELSE_ROOT (*robot.running.model.TryBranch attribute*), 563
IF_ELSE_ROOT (*robot.running.model.While attribute*), 559
IfBranch (*class in robot.model.control*), 241
IfBranch (*class in robot.result.model*), 474
IfBranch (*class in robot.running.model*), 560
IfBuilder (*class in robot.running.builder.transformers*), 546
IfElseHeader (*class in robot.parsing.model.statements*), 372
IfHandler (*class in robot.result.xmlelementhandlers*), 521
IfHeader (*class in robot.parsing.model.statements*), 373
IfHeaderLexer (*class in robot.parsing.lexer.statementlexers*), 331
IfLexer (*class in robot.parsing.lexer.blocklexers*), 322
IfParser (*class in robot.parsing.parser.blockparsers*), 390
IfRunner (*class in robot.running.bodyrunner*), 550
ignored_dirs (*robot.parsing.suitestructure.SuiteStructureBuilder attribute*), 393
ignored_prefixes (*robot.parsing.suitestructure.SuiteStructureBuilder attribute*), 393
imag (*robot.reporting.stringcache.StringIndex attribute*), 401
image_names() (*robot.libraries.dialogs_py.InputDialog method*), 177
image_names() (*robot.libraries.dialogs_py.MessageDialog method*), 163
image_names() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 205
image_names() (*robot.libraries.dialogs_py.PassFailDialog method*), 219
image_names() (*robot.libraries.dialogs_py.SelectionDialog method*), 191
image_types() (*robot.libraries.dialogs_py.InputDialog method*), 177
image_types() (*robot.libraries.dialogs_py.MessageDialog method*), 163
image_types() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 205
image_types() (*robot.libraries.dialogs_py.PassFailDialog method*), 219
image_types() (*robot.libraries.dialogs_py.SelectionDialog method*), 191
ImplicitCommentLexer (*class in robot.parsing.lexer.statementlexers*), 330
ImplicitCommentSectionLexer (*class in robot.parsing.lexer.blocklexers*), 320
ImplicitCommentSectionParser (*class in robot.parsing.parser.fileparser*), 391
Import (*class in robot.running.model*), 576
import_() (*robot.utils.importer.ByPathImporter method*), 601

import_() (*robot.utils.importer.DottedImporter method*), 602
import_() (*robot.utils.importer.NonDottedImporter method*), 602
import_class_or_module() (*robot.utils.importer.Importer method*), 600
import_class_or_module_by_path() (*robot.utils.importer.Importer method*), 601
import_library() (*robot.libraries.BuiltIn.BuiltIn method*), 82
import_library() (*robot.running.importer.Importer method*), 552
import_library() (*robot.running.namespace.Namespace method*), 577
import_listeners() (*robot.output.listeners.ListenerProxy class method*), 307
import_module() (*robot.utils.importer.Importer method*), 601
import_resource() (*robot.libraries.BuiltIn.BuiltIn method*), 82
import_resource() (*robot.running.importer.Importer method*), 552
import_resource() (*robot.running.namespace.Namespace method*), 577
import_variables() (*robot.libraries.BuiltIn.BuiltIn method*), 82
import_variables() (*robot.running.namespace.Namespace method*), 577
import_variables() (*robot.variables.filesetter.PythonImporter method*), 607
import_variables() (*robot.variables.filesetter.YamlImporter method*), 607
ImportCache (*class in robot.running.importer*), 552
imported() (*robot.output.listeners.LibraryListeners method*), 307
imported() (*robot.output.listeners.Listeners method*), 307
imported() (*robot.output.logger.Logger method*), 308
Importer (*class in robot.running.importer*), 552
Importer (*class in robot.utils.importer*), 600
Imports (*class in robot.running.model*), 576
imports (*robot.running.model.ResourceFile attribute*), 576
in_for (*robot.parsing.model.blocks.ValidationContext attribute*), 345
in_keyword (*robot.parsing.model.blocks.ValidationContext attribute*), 345
in_while (*robot.parsing.model.blocks.ValidationContext attribute*), 345
include (*robot.conf.settings.RebotSettings attribute*), 66
include (*robot.conf.settings.RobotSettings attribute*), 65
include_suites (*robot.model.filter.Filter attribute*), 254
include_tags (*robot.model.filter.Filter attribute*), 254
include_tests (*robot.model.filter.Filter attribute*), 254
index() (*robot.model.body.BaseBody method*), 230
index() (*robot.model.body.Body method*), 232
index() (*robot.model.body.Branches method*), 233
index() (*robot.model.itemlist.ItemList method*), 259
index() (*robot.model.keyword.Keywords method*), 262
index() (*robot.model.message.Messages method*), 264
index() (*robot.model.testcase.TestCases method*), 284
index() (*robot.model.testsuite.TestSuites method*), 288
index() (*robot.result.model.Body method*), 459
index() (*robot.result.model.Branches method*), 460
index() (*robot.result.model.Iterations method*), 462
index() (*robot.running.model.Body method*), 555
index() (*robot.running.model.Imports method*), 577
info (*robot.model.stats.CombinedTagStat attribute*), 276
info (*robot.model.stats.TagStat attribute*), 276
info() (*in module robot.api.logger*), 15
info() (*in module robot.output.librarylogger*), 305
info() (*robot.output.console.verbose.VerboseWriter method*), 304
info() (*robot.output.filelogger.FileLogger method*), 305
info() (*robot.output.logger.Logger method*), 309
info() (*robot.output.loggerhelper.AbstractLogger method*), 309
info() (*robot.output.output.Output method*), 311
info() (*robot.utils.application.DefaultLogger method*), 591
info() (*robot.utils.importer.NoLogger method*), 602
Information, 614
InitFileContext (*class in robot.parsing.lexer.context*), 324
InitFileSettings (*class in robot.parsing.lexer.settings*), 326
InitHandler() (*in module robot.running.handlers*), 551
inits (*robot.libdocpkg.model.LibraryDoc attribute*), 71
INLINE_IF (*robot.parsing.lexer.tokens.END attribute*), 339
INLINE_IF (*robot.parsing.lexer.tokens.EOS attribute*), 336
INLINE_IF (*robot.parsing.lexer.tokens.Token attribute*), 335

InlineIfHeader (*class in robot.parsing.model.statements*), 374
InlineIfHeaderLexer (*class in robot.parsing.lexer.statementlexers*), 331
InlineIfLexer (*class in robot.parsing.lexer.blocklexers*), 323
InlinePythonFinder (*class in robot.variables.finders*), 608
input() (*robot.parsing.lexer.blocklexers.BlockLexer method*), 318
input() (*robot.parsing.lexer.blocklexers.CommentSectionLexer method*), 320
input() (*robot.parsing.lexer.blocklexers.ErrorSectionLexer method*), 321
input() (*robot.parsing.lexer.blocklexers.FileLexer method*), 319
input() (*robot.parsing.lexer.blocklexers.ForLexer method*), 322
input() (*robot.parsing.lexer.blocklexers.IfLexer method*), 322
input() (*robot.parsing.lexer.blocklexers.ImplicitCommentSectionLexer method*), 321
input() (*robot.parsing.lexer.blocklexers.InlineIfLexer method*), 323
input() (*robot.parsing.lexer.blocklexers.KeywordLexer method*), 322
input() (*robot.parsing.lexer.blocklexers.KeywordSectionLexer method*), 320
input() (*robot.parsing.lexer.blocklexers.NestedBlockLexer method*), 322
input() (*robot.parsing.lexer.blocklexers.SectionLexer method*), 319
input() (*robot.parsing.lexer.blocklexers.SettingSectionLexer method*), 319
input() (*robot.parsing.lexer.blocklexers.TaskSectionLexer method*), 320
input() (*robot.parsing.lexer.blocklexers.TestCaseLexer method*), 321
input() (*robot.parsing.lexer.blocklexers.TestCaseSectionLexer method*), 320
input() (*robot.parsing.lexer.blocklexers.TestOrKeywordLexer method*), 321
input() (*robot.parsing.lexer.blocklexers.TryLexer method*), 323
input() (*robot.parsing.lexer.blocklexers.VariableSectionLexer method*), 319
input() (*robot.parsing.lexer.blocklexers.WhileLexer method*), 322
input() (*robot.parsing.lexer.lexer.Lexer method*), 326
input() (*robot.parsing.lexer.statementlexers.BreakLexer method*), 333
input() (*robot.parsing.lexer.statementlexers.CommentLexer method*), 330
input() (*robot.parsing.lexer.statementlexers.CommentSectionHeaderLexer method*), 329
input() (*robot.parsing.lexer.statementlexers.ContinueLexer method*), 333
input() (*robot.parsing.lexer.statementlexers.ElseHeaderLexer method*), 331
input() (*robot.parsing.lexer.statementlexers.ElseIfHeaderLexer method*), 331
input() (*robot.parsing.lexer.statementlexers.EndLexer method*), 332
input() (*robot.parsing.lexer.statementlexers.ErrorSectionHeaderLexer method*), 329
input() (*robot.parsing.lexer.statementlexers.ExceptHeaderLexer method*), 332
input() (*robot.parsing.lexer.statementlexers.FinallyHeaderLexer method*), 332
input() (*robot.parsing.lexer.statementlexers.ForHeaderLexer method*), 331
input() (*robot.parsing.lexer.statementlexers.IfHeaderLexer method*), 331
input() (*robot.parsing.lexer.statementlexers.ImplicitCommentLexer method*), 330
input() (*robot.parsing.lexer.statementlexers.InlineIfHeaderLexer method*), 331
input() (*robot.parsing.lexer.statementlexers.KeywordCallLexer method*), 330
input() (*robot.parsing.lexer.statementlexers.KeywordSectionHeaderLexer method*), 329
input() (*robot.parsing.lexer.statementlexers.Lexer method*), 327
input() (*robot.parsing.lexer.statementlexers.ReturnLexer method*), 333
input() (*robot.parsing.lexer.statementlexers.SectionHeaderLexer method*), 328
input() (*robot.parsing.lexer.statementlexers.SettingLexer method*), 330
input() (*robot.parsing.lexer.statementlexers.SettingSectionHeaderLexer method*), 328
input() (*robot.parsing.lexer.statementlexers.SingleType method*), 328
input() (*robot.parsing.lexer.statementlexers.StatementLexer method*), 328
input() (*robot.parsing.lexer.statementlexers.TaskSectionHeaderLexer method*), 329
input() (*robot.parsing.lexer.statementlexers.TestCaseSectionHeaderLexer method*), 329
input() (*robot.parsing.lexer.statementlexers.TestOrKeywordSettingLexer method*), 330
input() (*robot.parsing.lexer.statementlexers.TryHeaderLexer method*), 332
input() (*robot.parsing.lexer.statementlexers.TypeAndArguments method*), 328
input() (*robot.parsing.lexer.statementlexers.VariableLexer method*), 330
input() (*robot.parsing.lexer.statementlexers.VariableSectionHeaderLexer*

method), 328
input() (*robot.parsing.lexer.statementlexers.WhileHeaderLexer method*), 332
InputDialog (*class in robot.libraries.dialogs_py*), 171
insert() (*robot.model.body.BaseBody method*), 230
insert() (*robot.model.body.Body method*), 232
insert() (*robot.model.body.Branches method*), 233
insert() (*robot.model.itemlist.ItemList method*), 259
insert() (*robot.model.keyword.Keywords method*), 262
insert() (*robot.model.message.Messages method*), 264
insert() (*robot.model.testcase.TestCases method*), 284
insert() (*robot.model.testsuite.TestSuites method*), 288
insert() (*robot.result.model.Body method*), 459
insert() (*robot.result.model.Branches method*), 460
insert() (*robot.result.model.Iterations method*), 462
insert() (*robot.running.model.Body method*), 555
insert() (*robot.running.model.Imports method*), 577
insert_into_list() (*robot.libraries.Collections.Collections method*), 104
IntegerConverter (*class in robot.running.arguments.typeconverters*), 533
IntegerDumper (*class in robot.htmldata.jsonwriter*), 68
interact() (*robot.libraries.Telnet.TelnetConnection method*), 145
INTERNAL_UPDATE_FREQUENCY (*robot.libraries.Telnet.TelnetConnection attribute*), 141
is_assign() (*in module robot.variables.search*), 611
is_assign() (*robot.variables.search.VariableMatch method*), 611
is_dict_assign() (*in module robot.variables.search*), 611
is_dict_assign() (*robot.variables.search.VariableMatch method*), 611
is_dict_variable() (*in module robot.variables.search*), 611
is_dict_variable() (*robot.variables.search.VariableMatch method*), 611
is_directory (*robot.parsing.suitestructure.SuiteStructure attribute*), 392
is_global (*robot.running.libraryscopes.GlobalScope attribute*), 553
is_global (*robot.running.libraryscopes.TestCaseScope attribute*), 553
is_global (*robot.running.libraryscopes.TestSuiteScope attribute*), 553
is_list_assign() (*in module robot.variables.search*), 611
is_list_assign() (*robot.variables.search.VariableMatch method*), 611
is_list_variable() (*in module robot.variables.search*), 611
is_list_variable() (*robot.variables.search.VariableMatch method*), 611
is_process_running() (*robot.libraries.Process.Process method*), 125
is_scalar_assign() (*in module robot.variables.search*), 611
is_scalar_assign() (*robot.variables.search.VariableMatch method*), 611
is_scalar_variable() (*in module robot.variables.search*), 611
is_scalar_variable() (*robot.variables.search.VariableMatch method*), 611
is_variable() (*in module robot.variables.search*), 611
is_variable() (*robot.variables.search.VariableMatch method*), 611
isatty() (*in module robot.utils.misc*), 604
isatty() (*in module robot.utils.platform*), 605
IsLogged (*class in robot.output.loggerhelper*), 311
It (*class in robot.conf.languages*), 62
ItemList (*class in robot.model.itemlist*), 259
items() (*robot.model.metadata.Metadata method*), 264
items() (*robot.utils.dotdict.DotDict method*), 596
items() (*robot.utils.normalizing.NormalizedDict method*), 605
items() (*robot.variables.evaluation.EvaluationNamespace method*), 607
ITERATION (*robot.model.body.BodyItem attribute*), 228
ITERATION (*robot.model.control.Break attribute*), 249
ITERATION (*robot.model.control.Continue attribute*), 248
ITERATION (*robot.model.control.For attribute*), 239
ITERATION (*robot.model.control.If attribute*), 243
ITERATION (*robot.model.control.IfBranch attribute*), 242
ITERATION (*robot.model.control.Return attribute*), 247
ITERATION (*robot.model.control.Try attribute*), 246
ITERATION (*robot.model.control.TryBranch attribute*), 244
ITERATION (*robot.model.control.While attribute*), 240
ITERATION (*robot.model.keyword.Keyword attribute*), 260

ITERATION (*robot.model.message.Message attribute*), 263
ITERATION (*robot.output.loggerhelper.Message attribute*), 310
ITERATION (*robot.result.model.Break attribute*), 488
ITERATION (*robot.result.model.Continue attribute*), 486
ITERATION (*robot.result.model.For attribute*), 467
ITERATION (*robot.result.model.ForIteration attribute*), 465
ITERATION (*robot.result.model.If attribute*), 476
ITERATION (*robot.result.model.IfBranch attribute*), 474
ITERATION (*robot.result.model.Keyword attribute*), 491
ITERATION (*robot.result.model.Message attribute*), 463
ITERATION (*robot.result.model.Return attribute*), 483
ITERATION (*robot.result.model.Try attribute*), 481
ITERATION (*robot.result.model.TryBranch attribute*), 479
ITERATION (*robot.result.model.While attribute*), 472
ITERATION (*robot.result.model.WhileIteration attribute*), 470
ITERATION (*robot.running.model.Break attribute*), 569
ITERATION (*robot.running.model.Continue attribute*), 568
ITERATION (*robot.running.model.For attribute*), 558
ITERATION (*robot.running.model.If attribute*), 562
ITERATION (*robot.running.model.IfBranch attribute*), 561
ITERATION (*robot.running.model.Keyword attribute*), 556
ITERATION (*robot.running.model.Return attribute*), 566
ITERATION (*robot.running.model.Try attribute*), 565
ITERATION (*robot.running.model.TryBranch attribute*), 563
ITERATION (*robot.running.model.While attribute*), 559
iteration_class (*robot.result.model.For attribute*), 467
iteration_class (*robot.result.model.Iterations attribute*), 461
iteration_class (*robot.result.model.While attribute*), 471
IterationCountLimit (*class in robot.running.bodyrunner*), 550
IterationHandler (*class in robot.result.xmlelementhandlers*), 521
Iterations (*class in robot.result.model*), 461
iterations_class (*robot.result.model.For attribute*), 467
iterations_class (*robot.result.model.While attribute*), 471

J

join_command_line() (*robot.libraries.Process.Process method*), 128
join_path() (*robot.libraries.OperatingSystem.OperatingSystem method*), 119
join_paths() (*robot.libraries.OperatingSystem.OperatingSystem method*), 120
js_result (*robot.reporting.resultwriter.Results attribute*), 401
JsBuildingContext (*class in robot.reporting.jsbuildingcontext*), 393
JsExecutionResult (*class in robot.reporting.jsexecutionresult*), 394
JsFileWriter (*class in robot.htmldata.htmlfilewriter*), 67
JsModelBuilder (*class in robot.reporting.jsmodelbuilders*), 394
JsonConverter (*class in robot.testdoc*), 624
JsonDocBuilder (*class in robot.libdocpkg.jsonbuilder*), 71
JsonDumper (*class in robot.htmldata.jsonwriter*), 68
JsonWriter (*class in robot.htmldata.jsonwriter*), 68
JsResultWriter (*class in robot.reporting.jswriter*), 395

K

keep_in_dictionary() (*robot.libraries.Collections.Collections method*), 104
keys() (*robot.libraries.dialogs_py.InputDialog method*), 177
keys() (*robot.libraries.dialogs_py.MessageDialog method*), 163
keys() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 205
keys() (*robot.libraries.dialogs_py.PassFailDialog method*), 219
keys() (*robot.libraries.dialogs_py.SelectionDialog method*), 191
keys() (*robot.model.metadata.Metadata method*), 264
keys() (*robot.utils.dotdict.DotDict method*), 596
keys() (*robot.utils.normalizing.NormalizedDict method*), 605
keys() (*robot.variables.evaluation.EvaluationNamespace method*), 607
Keyword (*class in robot.model.keyword*), 259
Keyword (*class in robot.parsing.model.blocks*), 343
Keyword (*class in robot.result.model*), 490
Keyword (*class in robot.running.model*), 555
KEYWORD (*robot.model.body.BodyItem attribute*), 228
KEYWORD (*robot.model.control.Break attribute*), 249
KEYWORD (*robot.model.control.Continue attribute*), 248
KEYWORD (*robot.model.control.For attribute*), 239
KEYWORD (*robot.model.control.If attribute*), 243
KEYWORD (*robot.model.control.IfBranch attribute*), 242

KEYWORD (*robot.model.control.Return attribute*), 247
KEYWORD (*robot.model.control.Try attribute*), 246
KEYWORD (*robot.model.control.TryBranch attribute*), 244
KEYWORD (*robot.model.control.While attribute*), 240
KEYWORD (*robot.model.keyword.Keyword attribute*), 260
KEYWORD (*robot.model.message.Message attribute*), 263
KEYWORD (*robot.output.loggerhelper.Message attribute*), 310
KEYWORD (*robot.parsing.lexer.tokens.END attribute*), 339
KEYWORD (*robot.parsing.lexer.tokens.EOS attribute*), 336
KEYWORD (*robot.parsing.lexer.tokens.Token attribute*), 334
keyword (*robot.parsing.model.statements.KeywordCall attribute*), 370
KEYWORD (*robot.result.model.Break attribute*), 488
KEYWORD (*robot.result.model.Continue attribute*), 486
KEYWORD (*robot.result.model.For attribute*), 467
KEYWORD (*robot.result.model.ForIteration attribute*), 465
KEYWORD (*robot.result.model.If attribute*), 477
KEYWORD (*robot.result.model.IfBranch attribute*), 474
KEYWORD (*robot.result.model.Keyword attribute*), 491
KEYWORD (*robot.result.model.Message attribute*), 463
KEYWORD (*robot.result.model.Return attribute*), 483
KEYWORD (*robot.result.model.Try attribute*), 481
KEYWORD (*robot.result.model.TryBranch attribute*), 479
KEYWORD (*robot.result.model.While attribute*), 472
KEYWORD (*robot.result.model.WhileIteration attribute*), 470
KEYWORD (*robot.running.model.Break attribute*), 569
KEYWORD (*robot.running.model.Continue attribute*), 568
KEYWORD (*robot.running.model.For attribute*), 558
KEYWORD (*robot.running.model.If attribute*), 562
KEYWORD (*robot.running.model.IfBranch attribute*), 561
KEYWORD (*robot.running.model.Keyword attribute*), 556
KEYWORD (*robot.running.model.Return attribute*), 566
KEYWORD (*robot.running.model.Try attribute*), 565
KEYWORD (*robot.running.model.TryBranch attribute*), 563
KEYWORD (*robot.running.model.While attribute*), 559
keyword() (*in module robot.api.deco*), 11
keyword_class (*robot.model.body.BaseBody attribute*), 229
keyword_class (*robot.model.body.Body attribute*), 232
keyword_class (*robot.model.body.Branches attribute*), 233
keyword_class (*robot.result.model.Body attribute*), 459
keyword_class (*robot.result.model.Branches attribute*), 460
keyword_class (*robot.result.model.Iterations attribute*), 462
keyword_class (*robot.running.model.Body attribute*), 555
keyword_context() (*robot.parsing.lexer.context.FileContext method*), 323
keyword_context() (*robot.parsing.lexer.context.InitFileContext method*), 324
keyword_context() (*robot.parsing.lexer.context.ResourceFileContext method*), 324
keyword_context() (*robot.parsing.lexer.context.TestCaseFileContext method*), 324
KEYWORD_HEADER (*robot.parsing.lexer.tokens.END attribute*), 339
KEYWORD_HEADER (*robot.parsing.lexer.tokens.EOS attribute*), 337
KEYWORD_HEADER (*robot.parsing.lexer.tokens.Token attribute*), 334
keyword_marker() (*robot.output.console.verbose.VerboseWriter method*), 304
KEYWORD_NAME (*robot.parsing.lexer.tokens.END attribute*), 339
KEYWORD_NAME (*robot.parsing.lexer.tokens.EOS attribute*), 337
KEYWORD_NAME (*robot.parsing.lexer.tokens.Token attribute*), 334
keyword_section() (*robot.parsing.lexer.context.FileContext method*), 323
keyword_section() (*robot.parsing.lexer.context.InitFileContext method*), 324
keyword_section() (*robot.parsing.lexer.context.ResourceFileContext method*), 324
keyword_section() (*robot.parsing.lexer.context.TestCaseFileContext method*), 324
keyword_should_exist() (*robot.libraries.BuiltIn.BuiltIn method*), 82
KEYWORD_TAGS (*robot.parsing.lexer.tokens.END attribute*), 339
KEYWORD_TAGS (*robot.parsing.lexer.tokens.EOS attribute*), 337
KEYWORD_TAGS (*robot.parsing.lexer.tokens.Token attribute*), 334
keyword_tags_setting (*robot.conf.languages.Bg attribute*), 60
keyword_tags_setting (*robot.conf.languages.Bs*

attribute), 39
keyword_tags_setting (*robot.conf.languages.Cs attribute*), 36
keyword_tags_setting (*robot.conf.languages.De attribute*), 43
keyword_tags_setting (*robot.conf.languages.En attribute*), 35
keyword_tags_setting (*robot.conf.languages.Es attribute*), 51
keyword_tags_setting (*robot.conf.languages.Fi attribute*), 40
keyword_tags_setting (*robot.conf.languages.Fr attribute*), 41
keyword_tags_setting (*robot.conf.languages.Hi attribute*), 64
keyword_tags_setting (*robot.conf.languages.It attribute*), 62
keyword_tags_setting (*robot.conf.languages.Language attribute*), 33
keyword_tags_setting (*robot.conf.languages.Nl attribute*), 37
keyword_tags_setting (*robot.conf.languages.Pl attribute*), 48
keyword_tags_setting (*robot.conf.languages.Pt attribute*), 46
keyword_tags_setting (*robot.conf.languages.PtBr attribute*), 44
keyword_tags_setting (*robot.conf.languages.Ro attribute*), 61
keyword_tags_setting (*robot.conf.languages.Ru attribute*), 53
keyword_tags_setting (*robot.conf.languages.Sv attribute*), 58
keyword_tags_setting (*robot.conf.languages.Th attribute*), 47
keyword_tags_setting (*robot.conf.languages.Tr attribute*), 57
keyword_tags_setting (*robot.conf.languages.Uk attribute*), 50
keyword_tags_setting (*robot.conf.languages.ZhCn attribute*), 54
keyword_tags_setting (*robot.conf.languages.ZhTw attribute*), 55
keyword_timeout (*robot.errors.TimeoutError attribute*), 614
KeywordBuilder (*class in robot.reporting.jsmodelbuilders*), 394
KeywordBuilder (*class in robot.running.builder.transformers*), 545
KeywordCall (*class in robot.parsing.model.statements*), 370
KeywordCallLexer (*class in robot.parsing.lexer.statementlexers*), 330
KeywordCallTemplate (*class in robot.running.arguments.argumentmapper*), 528
KeywordContext (*class in robot.parsing.lexer.context*), 325
KeywordDoc (*class in robot.libdocpkg.model*), 72
KeywordDocBuilder (*class in robot.libdocpkg.robotbuilder*), 72
KeywordError, 614
KeywordHandler (*class in robot.result.xmlelementhandlers*), 520
KeywordLexer (*class in robot.parsing.lexer.blocklexers*), 321
KeywordMarker (*class in robot.output.console.verbose*), 304
KeywordMatcher (*class in robot.libdocpkg.consoleviewer*), 70
KeywordName (*class in robot.parsing.model.statements*), 363
KeywordParser (*class in robot.parsing.parser.blockparsers*), 389
KeywordRecommendationFinder (*class in robot.running.namespace*), 578
KeywordRemover() (*in module robot.result.keywordremover*), 415
KeywordRunner (*class in robot.running.bodyrunner*), 549
Keywords (*class in robot.model.keyword*), 261
keywords (*robot.libdocpkg.model.LibraryDoc attribute*), 71
keywords (*robot.model.control.For attribute*), 239
keywords (*robot.model.testcase.TestCase attribute*), 283
keywords (*robot.model.testsuite.TestSuite attribute*), 286
keywords (*robot.result.model.For attribute*), 468
keywords (*robot.result.model.Keyword attribute*), 490
keywords (*robot.result.model.TestCase attribute*), 494
keywords (*robot.result.model.TestSuite attribute*), 497
keywords (*robot.running.model.For attribute*), 559
keywords (*robot.running.model.ResourceFile attribute*), 576
keywords (*robot.running.model.TestCase attribute*), 571
keywords (*robot.running.model.TestSuite attribute*), 575
keywords (*robot.running.model.UserKeyword attribute*), 576
keywords_header (*robot.conf.languages.Bg attribute*), 59
keywords_header (*robot.conf.languages.Bs attribute*), 38
keywords_header (*robot.conf.languages.Cs attribute*), 35

keywords_header (*robot.conf.languages.De attribute*), 42
keywords_header (*robot.conf.languages.En attribute*), 34
keywords_header (*robot.conf.languages.Es attribute*), 51
keywords_header (*robot.conf.languages.Fi attribute*), 40
keywords_header (*robot.conf.languages.Fr attribute*), 41
keywords_header (*robot.conf.languages.Hi attribute*), 63
keywords_header (*robot.conf.languages.It attribute*), 62
keywords_header (*robot.conf.languages.Language attribute*), 32
keywords_header (*robot.conf.languages.Nl attribute*), 37
keywords_header (*robot.conf.languages.Pl attribute*), 48
keywords_header (*robot.conf.languages.Pt attribute*), 45
keywords_header (*robot.conf.languages.PtBr attribute*), 44
keywords_header (*robot.conf.languages.Ro attribute*), 60
keywords_header (*robot.conf.languages.Ru attribute*), 52
keywords_header (*robot.conf.languages.Sv attribute*), 58
keywords_header (*robot.conf.languages.Th attribute*), 46
keywords_header (*robot.conf.languages.Tr attribute*), 56
keywords_header (*robot.conf.languages.Uk attribute*), 49
keywords_header (*robot.conf.languages.ZhCn attribute*), 53
keywords_header (*robot.conf.languages.ZhTw attribute*), 55
KeywordSection (*class in robot.parsing.model.blocks*), 342
KeywordSectionHeaderLexer (*class in robot.parsing.lexer.statementlexers*), 329
KeywordSectionLexer (*class in robot.parsing.lexer.blocklexers*), 320
KeywordSectionParser (*class in robot.parsing.parser.fileparser*), 391
KeywordSettings (*class in robot.parsing.lexer.settings*), 327
KeywordStore (*class in robot.running.namespace*), 578
KeywordTags (*class in robot.parsing.model.statements*), 356
KeywordTimeout (*class in robot.running.timeouts*), 548
KILL_TIMEOUT (*robot.libraries.Process.Process attribute*), 125
kwname (*robot.result.model.Break attribute*), 489
kwname (*robot.result.model.Continue attribute*), 487
kwname (*robot.result.model.For attribute*), 468
kwname (*robot.result.model.ForIteration attribute*), 466
kwname (*robot.result.model.If attribute*), 478
kwname (*robot.result.model.IfBranch attribute*), 475
kwname (*robot.result.model.Keyword attribute*), 490
kwname (*robot.result.model.Return attribute*), 484
kwname (*robot.result.model.Try attribute*), 482
kwname (*robot.result.model.TryBranch attribute*), 480
kwname (*robot.result.model.While attribute*), 473
kwname (*robot.result.model.WhileIteration attribute*), 471
kwname (*robot.result.modeldeprecation.DeprecatedAttributesMixin attribute*), 500

L

Language (*class in robot.conf.languages*), 32
language (*robot.parsing.model.statements.Config attribute*), 386
Languages (*class in robot.conf.languages*), 32
languages (*robot.conf.settings.RobotSettings attribute*), 65
LastStatementFinder (*class in robot.parsing.model.blocks*), 345
length_should_be() (*robot.libraries.BuiltIn.BuiltIn method*), 83
level (*robot.model.message.Message attribute*), 262
level (*robot.output.loggerhelper.Message attribute*), 310
level (*robot.result.model.Message attribute*), 464
lex() (*robot.parsing.lexer.blocklexers.BlockLexer method*), 318
lex() (*robot.parsing.lexer.blocklexers.CommentSectionLexer method*), 320
lex() (*robot.parsing.lexer.blocklexers.ErrorSectionLexer method*), 321
lex() (*robot.parsing.lexer.blocklexers.FileLexer method*), 319
lex() (*robot.parsing.lexer.blocklexers.ForLexer method*), 322
lex() (*robot.parsing.lexer.blocklexers.IfLexer method*), 322
lex() (*robot.parsing.lexer.blocklexers.ImplicitCommentSectionLexer method*), 321
lex() (*robot.parsing.lexer.blocklexers.InlineIfLexer method*), 323
lex() (*robot.parsing.lexer.blocklexers.KeywordLexer method*), 322

lex() (*robot.parsing.lexer.blocklexers.KeywordSectionLexer method*), 320
lex() (*robot.parsing.lexer.blocklexers.NestedBlockLexer method*), 322
lex() (*robot.parsing.lexer.blocklexers.SectionLexer method*), 319
lex() (*robot.parsing.lexer.blocklexers.SettingSectionLexer method*), 319
lex() (*robot.parsing.lexer.blocklexers.TaskSectionLexer method*), 320
lex() (*robot.parsing.lexer.blocklexers.TestCaseLexer method*), 321
lex() (*robot.parsing.lexer.blocklexers.TestCaseSectionLexer method*), 320
lex() (*robot.parsing.lexer.blocklexers.TestOrKeywordLexer method*), 321
lex() (*robot.parsing.lexer.blocklexers.TryLexer method*), 323
lex() (*robot.parsing.lexer.blocklexers.VariableSectionLexer method*), 319
lex() (*robot.parsing.lexer.blocklexers.WhileLexer method*), 322
lex() (*robot.parsing.lexer.settings.InitFileSettings method*), 326
lex() (*robot.parsing.lexer.settings.KeywordSettings method*), 327
lex() (*robot.parsing.lexer.settings.ResourceFileSettings method*), 327
lex() (*robot.parsing.lexer.settings.Settings method*), 326
lex() (*robot.parsing.lexer.settings.TestCaseFileSettings method*), 326
lex() (*robot.parsing.lexer.settings.TestCaseSettings method*), 327
lex() (*robot.parsing.lexer.statementlexers.BreakLexer method*), 333
lex() (*robot.parsing.lexer.statementlexers.CommentLexer method*), 330
lex() (*robot.parsing.lexer.statementlexers.CommentSectionHeaderLexer method*), 329
lex() (*robot.parsing.lexer.statementlexers.ContinueLexer method*), 333
lex() (*robot.parsing.lexer.statementlexers.ElseHeaderLexer method*), 331
lex() (*robot.parsing.lexer.statementlexers.ElseIfHeaderLexer method*), 331
lex() (*robot.parsing.lexer.statementlexers.EndLexer method*), 332
lex() (*robot.parsing.lexer.statementlexers.ErrorSectionHeaderLexer method*), 329
lex() (*robot.parsing.lexer.statementlexers.ExceptHeaderLexer method*), 332
lex() (*robot.parsing.lexer.statementlexers.FinallyHeaderLexer method*), 332
lex() (*robot.parsing.lexer.statementlexers.ForHeaderLexer method*), 331
lex() (*robot.parsing.lexer.statementlexers.IfHeaderLexer method*), 331
lex() (*robot.parsing.lexer.statementlexers.ImplicitCommentLexer method*), 330
lex() (*robot.parsing.lexer.statementlexers.InlineIfHeaderLexer method*), 331
lex() (*robot.parsing.lexer.statementlexers.KeywordCallLexer method*), 330
lex() (*robot.parsing.lexer.statementlexers.KeywordSectionHeaderLexer method*), 329
lex() (*robot.parsing.lexer.statementlexers.Lexer method*), 327
lex() (*robot.parsing.lexer.statementlexers.ReturnLexer method*), 333
lex() (*robot.parsing.lexer.statementlexers.SectionHeaderLexer method*), 328
lex() (*robot.parsing.lexer.statementlexers.SettingLexer method*), 330
lex() (*robot.parsing.lexer.statementlexers.SettingSectionHeaderLexer method*), 328
lex() (*robot.parsing.lexer.statementlexers.SingleType method*), 328
lex() (*robot.parsing.lexer.statementlexers.StatementLexer method*), 328
lex() (*robot.parsing.lexer.statementlexers.TaskSectionHeaderLexer method*), 329
lex() (*robot.parsing.lexer.statementlexers.TestCaseSectionHeaderLexer method*), 329
lex() (*robot.parsing.lexer.statementlexers.TestOrKeywordSettingLexer method*), 330
lex() (*robot.parsing.lexer.statementlexers.TryHeaderLexer method*), 332
lex() (*robot.parsing.lexer.statementlexers.TypeAndArguments method*), 328
lex() (*robot.parsing.lexer.statementlexers.VariableLexer method*), 330
lex() (*robot.parsing.lexer.statementlexers.VariableSectionHeaderLexer method*), 328
lex() (*robot.parsing.lexer.statementlexers.WhileHeaderLexer method*), 332
lex_invalid_section() (*robot.parsing.lexer.context.FileContext method*), 323
lex_invalid_section() (*robot.parsing.lexer.context.InitFileContext method*), 324
lex_invalid_section() (*robot.parsing.lexer.context.ResourceFileContext method*), 324
lex_invalid_section() (*robot.parsing.lexer.context.TestCaseFileContext method*), 324

lex_setting() (*robot.parsing.lexer.context.FileContext method*), 323
lex_setting() (*robot.parsing.lexer.context.InitFileContext method*), 325
lex_setting() (*robot.parsing.lexer.context.KeywordContext method*), 325
lex_setting() (*robot.parsing.lexer.context.LexingContext method*), 323
lex_setting() (*robot.parsing.lexer.context.ResourceFileContext method*), 324
lex_setting() (*robot.parsing.lexer.context.TestCaseContext method*), 325
lex_setting() (*robot.parsing.lexer.context.TestCaseFileContext method*), 324
Lexer (*class in robot.parsing.lexer.lexer*), 326
Lexer (*class in robot.parsing.lexer.statementlexers*), 327
lexer_classes() (*robot.parsing.lexer.blocklexers.BlockLexer method*), 318
lexer_classes() (*robot.parsing.lexer.blocklexers.CommentSectionLexer method*), 320
lexer_classes() (*robot.parsing.lexer.blocklexers.ErrorSectionLexer method*), 321
lexer_classes() (*robot.parsing.lexer.blocklexers.FileLexer method*), 319
lexer_classes() (*robot.parsing.lexer.blocklexers.ForLexer method*), 322
lexer_classes() (*robot.parsing.lexer.blocklexers.IfLexer method*), 322
lexer_classes() (*robot.parsing.lexer.blocklexers.ImplicitCommentSectionLexer method*), 321
lexer_classes() (*robot.parsing.lexer.blocklexers.InlineIfLexer method*), 323
lexer_classes() (*robot.parsing.lexer.blocklexers.KeywordLexer method*), 322
lexer_classes() (*robot.parsing.lexer.blocklexers.KeywordSectionLexer method*), 320
lexer_classes() (*robot.parsing.lexer.blocklexers.NestedBlockLexer method*), 322
lexer_classes() (*robot.parsing.lexer.blocklexers.SectionLexer method*), 319
lexer_classes() (*robot.parsing.lexer.blocklexers.SettingSectionLexer method*), 319
lexer_classes() (*robot.parsing.lexer.blocklexers.TaskSectionLexer method*), 320
lexer_classes() (*robot.parsing.lexer.blocklexers.TestCaseLexer method*), 321
lexer_classes() (*robot.parsing.lexer.blocklexers.TestCaseSectionLexer method*), 320
lexer_classes() (*robot.parsing.lexer.blocklexers.TestOrKeywordLexer method*), 321
lexer_classes() (*robot.parsing.lexer.blocklexers.TryLexer method*), 323
lexer_classes() (*robot.parsing.lexer.blocklexers.VariableSectionLexer method*), 319
lexer_classes() (*robot.parsing.lexer.blocklexers.WhileLexer method*), 322
lexer_for() (*robot.parsing.lexer.blocklexers.BlockLexer method*), 318
lexer_for() (*robot.parsing.lexer.blocklexers.CommentSectionLexer method*), 320
lexer_for() (*robot.parsing.lexer.blocklexers.ErrorSectionLexer method*), 321
lexer_for() (*robot.parsing.lexer.blocklexers.FileLexer method*), 319
lexer_for() (*robot.parsing.lexer.blocklexers.ForLexer method*), 322
lexer_for() (*robot.parsing.lexer.blocklexers.IfLexer method*), 323
lexer_for() (*robot.parsing.lexer.blocklexers.ImplicitCommentSectionLexer method*), 321
lexer_for() (*robot.parsing.lexer.blocklexers.InlineIfLexer method*), 323
lexer_for() (*robot.parsing.lexer.blocklexers.KeywordLexer method*), 322
lexer_for() (*robot.parsing.lexer.blocklexers.KeywordSectionLexer method*), 320
lexer_for() (*robot.parsing.lexer.blocklexers.NestedBlockLexer method*), 322
lexer_for() (*robot.parsing.lexer.blocklexers.SectionLexer method*), 319
lexer_for() (*robot.parsing.lexer.blocklexers.SettingSectionLexer method*), 319
lexer_for() (*robot.parsing.lexer.blocklexers.TaskSectionLexer method*), 320
lexer_for() (*robot.parsing.lexer.blocklexers.TestCaseLexer method*), 321
lexer_for() (*robot.parsing.lexer.blocklexers.TestCaseSectionLexer method*), 320
lexer_for() (*robot.parsing.lexer.blocklexers.TestOrKeywordLexer method*), 321
lexer_for() (*robot.parsing.lexer.blocklexers.TryLexer method*), 323
lexer_for() (*robot.parsing.lexer.blocklexers.VariableSectionLexer method*), 319
lexer_for() (*robot.parsing.lexer.blocklexers.WhileLexer method*), 322
LexingContext (*class in robot.parsing.lexer.context*), 323
LibdocBuilder (*class in robot.libdoc*), 619
libdoc() (*in module robot.libdoc*), 619
LibdocCli (*class in robot.libdoc*), 619
LibdocHtmlWriter (*class in robot.libdocpkg.htmlwriter*), 71
LibdocJsonWriter (*class in robot.libdocpkg.jsonwriter*), 71
LibdocModelWriter (*class in robot.libdocpkg.htmlwriter*), 71
LibdocOutput (*class in robot.libdocpkg.output*), 72

LibdocWriter() (*in module robot.libdocpkg.writer*), 72
LibdocXmlWriter (*class in robot.libdocpkg.xmlwriter*), 73
libname (*robot.result.model.Break attribute*), 489
libname (*robot.result.model.Continue attribute*), 487
libname (*robot.result.model.For attribute*), 469
libname (*robot.result.model.ForIteration attribute*), 466
libname (*robot.result.model.If attribute*), 478
libname (*robot.result.model.IfBranch attribute*), 475
libname (*robot.result.model.Keyword attribute*), 490
libname (*robot.result.model.Return attribute*), 485
libname (*robot.result.model.Try attribute*), 482
libname (*robot.result.model.TryBranch attribute*), 480
libname (*robot.result.model.While attribute*), 473
libname (*robot.result.model.WhileIteration attribute*), 471
libname (*robot.result.modeldeprecation.DeprecatedAttributesMixin attribute*), 500
libname (*robot.running.librarykeywordrunner.EmbeddedArgumentsRunner attribute*), 552
libname (*robot.running.librarykeywordrunner.LibraryKeywordRunner attribute*), 552
libname (*robot.running.librarykeywordrunner.RunKeywordRunner attribute*), 552
libname (*robot.running.userkeywordrunner.EmbeddedArgumentsRunner attribute*), 590
libname (*robot.running.userkeywordrunner.UserKeywordRunner attribute*), 590
libraries (*robot.running.namespace.Namespace attribute*), 577
LIBRARY (*robot.parsing.lexer.tokens.END attribute*), 339
LIBRARY (*robot.parsing.lexer.tokens.EOS attribute*), 337
LIBRARY (*robot.parsing.lexer.tokens.Token attribute*), 334
library (*robot.running.handlers.EmbeddedArgumentsHandler attribute*), 551
library (*robot.running.librarykeywordrunner.EmbeddedArgumentsRunner attribute*), 552
library (*robot.running.librarykeywordrunner.LibraryKeywordRunner attribute*), 552
library (*robot.running.librarykeywordrunner.RunKeywordRunner attribute*), 552
library() (*in module robot.api.deco*), 11
library() (*robot.running.model.Imports method*), 577
library_setting (*robot.conf.languages.Bg attribute*), 59
library_setting (*robot.conf.languages.Bs attribute*), 38
library_setting (*robot.conf.languages.Cs attribute*), 35
library_setting (*robot.conf.languages.De attribute*), 42
library_setting (*robot.conf.languages.En attribute*), 34
library_setting (*robot.conf.languages.Es attribute*), 51
library_setting (*robot.conf.languages.Fi attribute*), 40
library_setting (*robot.conf.languages.Fr attribute*), 41
library_setting (*robot.conf.languages.Hi attribute*), 63
library_setting (*robot.conf.languages.It attribute*), 62
library_setting (*robot.conf.languages.Language attribute*), 32
library_setting (*robot.conf.languages.Nl attribute*), 37
library_setting (*robot.conf.languages.Pl attribute*), 48
library_setting (*robot.conf.languages.Pt attribute*), 45
library_setting (*robot.conf.languages.PtBr attribute*), 44
library_setting (*robot.conf.languages.Ro attribute*), 61
library_setting (*robot.conf.languages.Ru attribute*), 52
library_setting (*robot.conf.languages.Sv attribute*), 58
library_setting (*robot.conf.languages.Th attribute*), 47
library_setting (*robot.conf.languages.Tr attribute*), 56
library_setting (*robot.conf.languages.Uk attribute*), 49
library_setting (*robot.conf.languages.ZhCn attribute*), 54
library_setting (*robot.conf.languages.ZhTw attribute*), 55
LibraryDoc (*class in robot.libdocpkg.model*), 71
LibraryDocBuilder (*class in robot.libdocpkg.robotbuilder*), 72
LibraryDocumentation() (*in module robot.libdocpkg.builder*), 69
LibraryImport (*class in robot.parsing.model.statements*), 350
LibraryKeywordRunner (*class in robot.running.librarykeywordrunner*), 552
LibraryListenerMethods (*class in robot.output.listenermethods*), 307
LibraryListeners (*class in robot.output.listeners*), 307
LibraryScope() (*in module*

robot.running.libraryscopes), 553
lift() (*robot.libraries.dialogs_py.InputDialog method*), 177
lift() (*robot.libraries.dialogs_py.MessageDialog method*), 163
lift() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 205
lift() (*robot.libraries.dialogs_py.PassFailDialog method*), 219
lift() (*robot.libraries.dialogs_py.SelectionDialog method*), 191
limit (*robot.model.control.While attribute*), 240
limit (*robot.parsing.model.blocks.While attribute*), 344
limit (*robot.parsing.model.statements.WhileHeader attribute*), 381
limit (*robot.result.model.While attribute*), 473
limit (*robot.running.model.While attribute*), 560
limit_exceeded() (*robot.running.bodyrunner.DurationLimit method*), 550
limit_exceeded() (*robot.running.bodyrunner.IterationCountLimit method*), 550
limit_exceeded() (*robot.running.bodyrunner.NoLimit method*), 550
limit_exceeded() (*robot.running.bodyrunner.WhileLimit method*), 550
LineFormatter (*class in robot.utils.htmlformatters*), 599
lineno (*robot.model.testcase.TestCase attribute*), 283
lineno (*robot.parsing.lexer.tokens.END attribute*), 340
lineno (*robot.parsing.lexer.tokens.EOS attribute*), 338
lineno (*robot.parsing.lexer.tokens.Token attribute*), 335
lineno (*robot.parsing.model.blocks.Block attribute*), 340
lineno (*robot.parsing.model.blocks.CommentSection attribute*), 342
lineno (*robot.parsing.model.blocks.File attribute*), 341
lineno (*robot.parsing.model.blocks.For attribute*), 344
lineno (*robot.parsing.model.blocks.HeaderAndBody attribute*), 341
lineno (*robot.parsing.model.blocks.If attribute*), 343
lineno (*robot.parsing.model.blocks.Keyword attribute*), 343
lineno (*robot.parsing.model.blocks.KeywordSection attribute*), 342
lineno (*robot.parsing.model.blocks.Section attribute*), 341
lineno (*robot.parsing.model.blocks.SettingSection attribute*), 341
lineno (*robot.parsing.model.blocks.TestCase attribute*), 343
lineno (*robot.parsing.model.blocks.TestCaseSection attribute*), 342
lineno (*robot.parsing.model.blocks.Try attribute*), 344
lineno (*robot.parsing.model.blocks.VariableSection attribute*), 342
lineno (*robot.parsing.model.blocks.While attribute*), 344
lineno (*robot.parsing.model.statements.Arguments attribute*), 369
lineno (*robot.parsing.model.statements.Break attribute*), 385
lineno (*robot.parsing.model.statements.Comment attribute*), 386
lineno (*robot.parsing.model.statements.Config attribute*), 386
lineno (*robot.parsing.model.statements.Continue attribute*), 384
lineno (*robot.parsing.model.statements.DefaultTags attribute*), 356
lineno (*robot.parsing.model.statements.Documentation attribute*), 353
lineno (*robot.parsing.model.statements.DocumentationOrMetadata attribute*), 347
lineno (*robot.parsing.model.statements.ElseHeader attribute*), 377
lineno (*robot.parsing.model.statements.ElseIfHeader attribute*), 376
lineno (*robot.parsing.model.statements.EmptyLine attribute*), 388
lineno (*robot.parsing.model.statements.End attribute*), 381
lineno (*robot.parsing.model.statements.Error attribute*), 387
lineno (*robot.parsing.model.statements.ExceptHeader attribute*), 379
lineno (*robot.parsing.model.statements.FinallyHeader attribute*), 380
lineno (*robot.parsing.model.statements.Fixture attribute*), 349
lineno (*robot.parsing.model.statements.ForceTags attribute*), 355
lineno (*robot.parsing.model.statements.ForHeader attribute*), 372
lineno (*robot.parsing.model.statements.IfElseHeader attribute*), 373
lineno (*robot.parsing.model.statements.IfHeader attribute*), 374
lineno (*robot.parsing.model.statements.InlineIfHeader attribute*), 375
lineno (*robot.parsing.model.statements.KeywordCall attribute*), 371
lineno (*robot.parsing.model.statements.KeywordName attribute*), 364
lineno (*robot.parsing.model.statements.KeywordTags attribute*), 357
lineno (*robot.parsing.model.statements.LibraryImport attribute*), 351
lineno (*robot.parsing.model.statements.LoopControl*

attribute), 383
lineno (*robot.parsing.model.statements.Metadata attribute*), 354
lineno (*robot.parsing.model.statements.MultiValue attribute*), 348
lineno (*robot.parsing.model.statements.NoArgumentHeader attribute*), 377
lineno (*robot.parsing.model.statements.ResourceImport attribute*), 352
lineno (*robot.parsing.model.statements.Return attribute*), 370
lineno (*robot.parsing.model.statements.ReturnStatement attribute*), 382
lineno (*robot.parsing.model.statements.SectionHeader attribute*), 350
lineno (*robot.parsing.model.statements.Setup attribute*), 365
lineno (*robot.parsing.model.statements.SingleValue attribute*), 348
lineno (*robot.parsing.model.statements.Statement attribute*), 346
lineno (*robot.parsing.model.statements.SuiteSetup attribute*), 358
lineno (*robot.parsing.model.statements.SuiteTeardown attribute*), 358
lineno (*robot.parsing.model.statements.Tags attribute*), 367
lineno (*robot.parsing.model.statements.Teardown attribute*), 366
lineno (*robot.parsing.model.statements.Template attribute*), 367
lineno (*robot.parsing.model.statements.TemplateArguments attribute*), 372
lineno (*robot.parsing.model.statements.TestCaseName attribute*), 363
lineno (*robot.parsing.model.statements.TestSetup attribute*), 359
lineno (*robot.parsing.model.statements.TestTeardown attribute*), 360
lineno (*robot.parsing.model.statements.TestTemplate attribute*), 361
lineno (*robot.parsing.model.statements.TestTimeout attribute*), 362
lineno (*robot.parsing.model.statements.Timeout attribute*), 368
lineno (*robot.parsing.model.statements.TryHeader attribute*), 378
lineno (*robot.parsing.model.statements.Variable attribute*), 363
lineno (*robot.parsing.model.statements.VariablesImport attribute*), 353
lineno (*robot.parsing.model.statements.WhileHeader attribute*), 382
lineno (*robot.result.model.TestCase attribute*), 495
lineno (*robot.running.model.Break attribute*), 569
lineno (*robot.running.model.Continue attribute*), 567
lineno (*robot.running.model.For attribute*), 557
lineno (*robot.running.model.If attribute*), 562
lineno (*robot.running.model.IfBranch attribute*), 560
lineno (*robot.running.model.Keyword attribute*), 555
lineno (*robot.running.model.Return attribute*), 566
lineno (*robot.running.model.TestCase attribute*), 571
lineno (*robot.running.model.Try attribute*), 564
lineno (*robot.running.model.TryBranch attribute*), 563
lineno (*robot.running.model.While attribute*), 559
lines (*robot.parsing.model.statements.Arguments attribute*), 369
lines (*robot.parsing.model.statements.Break attribute*), 385
lines (*robot.parsing.model.statements.Comment attribute*), 386
lines (*robot.parsing.model.statements.Config attribute*), 387
lines (*robot.parsing.model.statements.Continue attribute*), 384
lines (*robot.parsing.model.statements.DefaultTags attribute*), 356
lines (*robot.parsing.model.statements.Documentation attribute*), 354
lines (*robot.parsing.model.statements.DocumentationOrMetadata attribute*), 347
lines (*robot.parsing.model.statements.ElseHeader attribute*), 377
lines (*robot.parsing.model.statements.ElseIfHeader attribute*), 376
lines (*robot.parsing.model.statements.EmptyLine attribute*), 388
lines (*robot.parsing.model.statements.End attribute*), 381
lines (*robot.parsing.model.statements.Error attribute*), 387
lines (*robot.parsing.model.statements.ExceptHeader attribute*), 379
lines (*robot.parsing.model.statements.FinallyHeader attribute*), 380
lines (*robot.parsing.model.statements.Fixture attribute*), 349
lines (*robot.parsing.model.statements.ForceTags attribute*), 355
lines (*robot.parsing.model.statements.ForHeader attribute*), 372
lines (*robot.parsing.model.statements.IfElseHeader attribute*), 373
lines (*robot.parsing.model.statements.IfHeader attribute*), 374
lines (*robot.parsing.model.statements.InlineIfHeader attribute*), 375
lines (*robot.parsing.model.statements.KeywordCall at-*

tribute), 371
lines (*robot.parsing.model.statements.KeywordName attribute*), 364
lines (*robot.parsing.model.statements.KeywordTags attribute*), 357
lines (*robot.parsing.model.statements.LibraryImport attribute*), 351
lines (*robot.parsing.model.statements.LoopControl attribute*), 383
lines (*robot.parsing.model.statements.Metadata attribute*), 354
lines (*robot.parsing.model.statements.MultiValue attribute*), 349
lines (*robot.parsing.model.statements.NoArgumentHeader attribute*), 377
lines (*robot.parsing.model.statements.ResourceImport attribute*), 352
lines (*robot.parsing.model.statements.Return attribute*), 370
lines (*robot.parsing.model.statements.ReturnStatement attribute*), 382
lines (*robot.parsing.model.statements.SectionHeader attribute*), 350
lines (*robot.parsing.model.statements.Setup attribute*), 365
lines (*robot.parsing.model.statements.SingleValue attribute*), 348
lines (*robot.parsing.model.statements.Statement attribute*), 346
lines (*robot.parsing.model.statements.SuiteSetup attribute*), 358
lines (*robot.parsing.model.statements.SuiteTeardown attribute*), 358
lines (*robot.parsing.model.statements.Tags attribute*), 367
lines (*robot.parsing.model.statements.Teardown attribute*), 366
lines (*robot.parsing.model.statements.Template attribute*), 367
lines (*robot.parsing.model.statements.TemplateArguments attribute*), 372
lines (*robot.parsing.model.statements.TestCaseName attribute*), 363
lines (*robot.parsing.model.statements.TestSetup attribute*), 359
lines (*robot.parsing.model.statements.TestTeardown attribute*), 360
lines (*robot.parsing.model.statements.TestTemplate attribute*), 361
lines (*robot.parsing.model.statements.TestTimeout attribute*), 362
lines (*robot.parsing.model.statements.Timeout attribute*), 368
lines (*robot.parsing.model.statements.TryHeader attribute*), 378
lines (*robot.parsing.model.statements.Variable attribute*), 363
lines (*robot.parsing.model.statements.VariablesImport attribute*), 353
lines (*robot.parsing.model.statements.WhileHeader attribute*), 382
LineWriter (*class in robot.htmldata.htmlfilewriter*), 67
link() (*robot.reporting.jsbuildingcontext.JsBuildingContext method*), 394
LinkFormatter (*class in robot.utils.htmlformatters*), 599
links (*robot.model.stats.TagStat attribute*), 276
list() (*robot.libdocpkg.consoleviewer.ConsoleViewer method*), 70
list_directories_in_directory() (*robot.libraries.OperatingSystem.OperatingSystem method*), 122
list_directory() (*robot.libraries.OperatingSystem.OperatingSystem method*), 121
list_files_in_directory() (*robot.libraries.OperatingSystem.OperatingSystem method*), 121
list_should_contain_sub_list() (*robot.libraries.Collections.Collections method*), 104
list_should_contain_value() (*robot.libraries.Collections.Collections method*), 104
list_should_not_contain_duplicates() (*robot.libraries.Collections.Collections method*), 104
list_should_not_contain_value() (*robot.libraries.Collections.Collections method*), 105
ListConverter (*class in robot.running.arguments.typeconverters*), 537
listener() (*robot.libraries.Telnet.TelnetConnection method*), 145
ListenerArguments (*class in robot.output.listenerarguments*), 306
ListenerMethod (*class in robot.output.listenermethods*), 307
ListenerMethods (*class in robot.output.listenermethods*), 307
ListenerProxy (*class in robot.output.listeners*), 307
Listeners (*class in robot.output.listeners*), 307
listeners (*robot.conf.settings.RobotSettings attribute*), 65
ListFormatter (*class in robot.utils.htmlformatters*), 600
lists_should_be_equal()

(*robot.libraries.Collections.Collections method*), 105
ListVariableTableValue (*class in robot.variables.tablesetter*), 612
Location (*class in robot.libraries.XML*), 157
log (*robot.conf.settings.RebotSettings attribute*), 66
log (*robot.conf.settings.RobotSettings attribute*), 65
log() (*robot.libraries.BuiltIn.BuiltIn method*), 83
log_config (*robot.conf.settings.RebotSettings attribute*), 66
log_dictionary() (*robot.libraries.Collections.Collections method*), 105
log_element() (*robot.libraries.XML.XML method*), 156
log_environment_variables() (*robot.libraries.OperatingSystem.OperatingSystem method*), 119
log_file() (*robot.libraries.OperatingSystem.OperatingSystem method*), 115
log_level (*robot.conf.settings.RebotSettings attribute*), 66
log_level (*robot.conf.settings.RobotSettings attribute*), 65
log_list() (*robot.libraries.Collections.Collections method*), 105
log_many() (*robot.libraries.BuiltIn.BuiltIn method*), 83
log_message() (*robot.output.listeners.LibraryListeners method*), 307
log_message() (*robot.output.listeners.Listeners method*), 307
log_message() (*robot.output.logger.Logger method*), 308
log_message() (*robot.output.xmllogger.XmlLogger method*), 313
log_message() (*robot.reporting.outputwriter.OutputWriter method*), 397
log_output() (*robot.output.logger.Logger method*), 308
log_to_console() (*robot.libraries.BuiltIn.BuiltIn method*), 83
log_variables() (*robot.libraries.BuiltIn.BuiltIn method*), 84
Logger (*class in robot.output.logger*), 308
LoggerProxy (*class in robot.output.logger*), 309
login() (*robot.libraries.Telnet.TelnetConnection method*), 143
LogWriter (*class in robot.reporting.logreportwriters*), 395
longname (*robot.model.testcase.TestCase attribute*), 284
longname (*robot.model.testsuite.TestSuite attribute*), 285
longname (*robot.result.model.TestCase attribute*), 495
longname (*robot.result.model.TestSuite attribute*), 497
longname (*robot.running.librarykeywordrunner.EmbeddedArgumentsRun attribute*), 552
longname (*robot.running.librarykeywordrunner.LibraryKeywordRunner attribute*), 552
longname (*robot.running.librarykeywordrunner.RunKeywordRunner attribute*), 552
longname (*robot.running.model.TestCase attribute*), 571
longname (*robot.running.model.TestSuite attribute*), 575
longname (*robot.running.usererrorhandler.UserErrorHandler attribute*), 589
longname (*robot.running.userkeyword.EmbeddedArgumentsHandler attribute*), 590
longname (*robot.running.userkeyword.UserKeywordHandler attribute*), 590
longname (*robot.running.userkeywordrunner.EmbeddedArgumentsRunner attribute*), 590
longname (*robot.running.userkeywordrunner.UserKeywordRunner attribute*), 590
LoopControl (*class in robot.parsing.model.statements*), 383
lower() (*robot.libraries.dialogs_py.InputDialog method*), 177
lower() (*robot.libraries.dialogs_py.MessageDialog method*), 163
lower() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 205
lower() (*robot.libraries.dialogs_py.PassFailDialog method*), 219
lower() (*robot.libraries.dialogs_py.SelectionDialog method*), 191

M

main() (*robot.libdoc.LibDoc method*), 619
main() (*robot.rebot.Rebot method*), 620
main() (*robot.run.RobotFramework method*), 622
main() (*robot.testdoc.TestDoc method*), 624
main() (*robot.utils.application.Application method*), 591
mainloop() (*robot.libraries.dialogs_py.InputDialog method*), 177
mainloop() (*robot.libraries.dialogs_py.MessageDialog method*), 163
mainloop() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 205
mainloop() (*robot.libraries.dialogs_py.PassFailDialog method*), 219
mainloop() (*robot.libraries.dialogs_py.SelectionDialog method*), 191
make_connection() (*robot.libraries.Remote.TimeoutHTTPSTransport method*), 129

make_connection() (*robot.libraries.Remote.TimeoutHTTPTransport method*), 129
manage() (*robot.libraries.dialogs_py.InputDialog method*), 177
manage() (*robot.libraries.dialogs_py.MessageDialog method*), 163
manage() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 205
manage() (*robot.libraries.dialogs_py.PassFailDialog method*), 219
manage() (*robot.libraries.dialogs_py.SelectionDialog method*), 191
map() (*robot.running.arguments.argumentmapper.ArgumentMapper method*), 528
map() (*robot.running.arguments.argumentspec.ArgumentSpec method*), 530
map() (*robot.running.arguments.embedded.EmbeddedArguments method*), 531
MappingDumper (*class in robot.htmldata.jsonwriter*), 68
mark() (*robot.output.console.verbose.KeywordMarker method*), 304
match (*robot.variables.search.VariableMatch attribute*), 611
match() (*robot.model.namepatterns.NamePatterns method*), 270
match() (*robot.model.namepatterns.SuiteNamePatterns method*), 270
match() (*robot.model.namepatterns.TestNamePatterns method*), 270
match() (*robot.model.stats.CombinedTagStat method*), 276
match() (*robot.model.tags.AndTagPattern method*), 277
match() (*robot.model.tags.NotTagPattern method*), 277
match() (*robot.model.tags.OrTagPattern method*), 277
match() (*robot.model.tags.SingleTagPattern method*), 277
match() (*robot.model.tags.TagPatterns method*), 277
match() (*robot.model.tags.Tags method*), 277
match() (*robot.model.tagstatistics.TagStatDoc method*), 282
match() (*robot.model.tagstatistics.TagStatLink method*), 282
match() (*robot.reporting.expandkeywordmatcher.ExpandKeywordMatcher method*), 393
match() (*robot.result.flattenkeywordmatcher.FlattenByNameMatcher method*), 415
match() (*robot.result.flattenkeywordmatcher.FlattenByTagMatcher method*), 415
match() (*robot.result.flattenkeywordmatcher.FlattenByTypeMatcher method*), 415
match() (*robot.running.arguments.embedded.EmbeddedArguments method*), 531
match() (*robot.utils.htmlformatters.HeaderFormatter method*), 599
match() (*robot.utils.htmlformatters.RulerFormatter method*), 599
match() (*robot.utils.match.Matcher method*), 603
match() (*robot.utils.match.MultiMatcher method*), 603
match_any() (*robot.utils.match.Matcher method*), 603
match_any() (*robot.utils.match.MultiMatcher method*), 603
Matcher (*class in robot.utils.match*), 603
matches() (*robot.running.handlers.EmbeddedArgumentsHandler method*), 551
matches() (*robot.running.userkeyword.EmbeddedArgumentsHandler method*), 590
max_assign_length (*robot.conf.settings.RobotSettings attribute*), 65
max_error_lines (*robot.conf.settings.RobotSettings attribute*), 65
maxargs (*robot.running.arguments.argumentspec.ArgumentSpec attribute*), 530
maxsize() (*robot.libraries.dialogs_py.InputDialog method*), 177
maxsize() (*robot.libraries.dialogs_py.MessageDialog method*), 163
maxsize() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 205
maxsize() (*robot.libraries.dialogs_py.PassFailDialog method*), 219
maxsize() (*robot.libraries.dialogs_py.SelectionDialog method*), 191
merge (*robot.conf.settings.RebotSettings attribute*), 66
merge() (*robot.result.merger.Merger method*), 449
Merger (*class in robot.result.merger*), 449
Message (*class in robot.model.message*), 262
Message (*class in robot.output.loggerhelper*), 309
Message (*class in robot.result.model*), 463
message (*robot.errors.BreakLoop attribute*), 618
message (*robot.errors.ContinueLoop attribute*), 617
message (*robot.errors.DataError attribute*), 613
message (*robot.errors.ExecutionFailed attribute*), 615
message (*robot.errors.ExecutionFailures attribute*), 616
message (*robot.errors.ExecutionPassed attribute*), 616
message (*robot.errors.ExecutionStatus attribute*), 615
message (*robot.errors.FrameworkError attribute*), 613
message (*robot.errors.HandlerExecutionFailed attribute*), 615
message (*robot.errors.Information attribute*), 614
message (*robot.errors.KeywordError attribute*), 614
message (*robot.errors.PassExecution attribute*), 617
message (*robot.errors.RemoteError attribute*), 618
message (*robot.errors.ReturnFromKeyword attribute*), 618
message (*robot.errors.RobotError attribute*), 613

message (*robot.errors.TimeoutError attribute*), 614
message (*robot.errors.UserKeywordExecutionFailed attribute*), 616
message (*robot.errors.VariableError attribute*), 614
MESSAGE (*robot.model.body.BodyItem attribute*), 228
MESSAGE (*robot.model.control.Break attribute*), 249
MESSAGE (*robot.model.control.Continue attribute*), 248
MESSAGE (*robot.model.control.For attribute*), 239
MESSAGE (*robot.model.control.If attribute*), 243
MESSAGE (*robot.model.control.IfBranch attribute*), 242
MESSAGE (*robot.model.control.Return attribute*), 247
MESSAGE (*robot.model.control.Try attribute*), 246
MESSAGE (*robot.model.control.TryBranch attribute*), 244
MESSAGE (*robot.model.control.While attribute*), 240
MESSAGE (*robot.model.keyword.Keyword attribute*), 260
MESSAGE (*robot.model.message.Message attribute*), 263
message (*robot.model.message.Message attribute*), 262
message (*robot.model.totalstatistics.TotalStatistics attribute*), 288
MESSAGE (*robot.output.loggerhelper.Message attribute*), 310
message (*robot.output.loggerhelper.Message attribute*), 309
MESSAGE (*robot.result.model.Break attribute*), 488
message (*robot.result.model.Break attribute*), 489
MESSAGE (*robot.result.model.Continue attribute*), 486
message (*robot.result.model.Continue attribute*), 487
MESSAGE (*robot.result.model.For attribute*), 467
message (*robot.result.model.For attribute*), 469
MESSAGE (*robot.result.model.ForIteration attribute*), 465
message (*robot.result.model.ForIteration attribute*), 466
MESSAGE (*robot.result.model.If attribute*), 477
message (*robot.result.model.If attribute*), 478
MESSAGE (*robot.result.model.IfBranch attribute*), 474
message (*robot.result.model.IfBranch attribute*), 475
MESSAGE (*robot.result.model.Keyword attribute*), 491
message (*robot.result.model.Keyword attribute*), 490
MESSAGE (*robot.result.model.Message attribute*), 463
message (*robot.result.model.Message attribute*), 464
MESSAGE (*robot.result.model.Return attribute*), 483
message (*robot.result.model.Return attribute*), 485
message (*robot.result.model.TestCase attribute*), 493
message (*robot.result.model.TestSuite attribute*), 496
MESSAGE (*robot.result.model.Try attribute*), 481
message (*robot.result.model.Try attribute*), 482
MESSAGE (*robot.result.model.TryBranch attribute*), 479
message (*robot.result.model.TryBranch attribute*), 480
MESSAGE (*robot.result.model.While attribute*), 472
message (*robot.result.model.While attribute*), 473
MESSAGE (*robot.result.model.WhileIteration attribute*), 470
message (*robot.result.model.WhileIteration attribute*), 471
message (*robot.result.modeldeprecation.DeprecatedAttributesMixin attribute*), 500
MESSAGE (*robot.running.model.Break attribute*), 569
MESSAGE (*robot.running.model.Continue attribute*), 568
MESSAGE (*robot.running.model.For attribute*), 558
MESSAGE (*robot.running.model.If attribute*), 562
MESSAGE (*robot.running.model.IfBranch attribute*), 561
MESSAGE (*robot.running.model.Keyword attribute*), 556
MESSAGE (*robot.running.model.Return attribute*), 566
MESSAGE (*robot.running.model.Try attribute*), 565
MESSAGE (*robot.running.model.TryBranch attribute*), 563
MESSAGE (*robot.running.model.While attribute*), 559
message (*robot.running.status.ParentMessage attribute*), 584
message (*robot.running.status.SuiteMessage attribute*), 584
message (*robot.running.status.SuiteStatus attribute*), 583
message (*robot.running.status.TestMessage attribute*), 584
message (*robot.running.status.TestStatus attribute*), 583
message (*robot.utils.error.ErrorDetails attribute*), 598
message() (*robot.output.console.dotted.DottedOutput method*), 299
message() (*robot.output.console.quiet.QuietOutput method*), 304
message() (*robot.output.console.verbose.VerboseOutput method*), 304
message() (*robot.output.console.verbose.VerboseWriter method*), 304
message() (*robot.output.filelogger.FileLogger method*), 305
message() (*robot.output.logger.Logger method*), 308
message() (*robot.output.loggerhelper.AbstractLogger method*), 309
message() (*robot.output.output.Output method*), 311
message() (*robot.output.xmllogger.XmlLogger method*), 313
message() (*robot.reporting.outputwriter.OutputWriter method*), 397
message_class (*robot.model.body.BaseBody attribute*), 229
message_class (*robot.model.body.Body attribute*), 232
message_class (*robot.model.body.Branches attribute*), 233
message_class (*robot.result.model.Body attribute*), 459
message_class (*robot.result.model.Branches attribute*), 461

message_class (*robot.result.model.Iterations attribute*), 462
message_class (*robot.running.model.Body attribute*), 555
message_level() (*robot.reporting.jsbuildingcontext.JsBuildingContext method*), 393
MessageArguments (*class in robot.output.listenerarguments*), 306
MessageBuilder (*class in robot.reporting.jsmodelbuilders*), 394
MessageDialog (*class in robot.libraries.dialogs_py*), 157
MessageFilter (*class in robot.result.messagefilter*), 453
MessageHandler (*class in robot.result.xmlelementhandlers*), 523
Messages (*class in robot.model.message*), 264
messages (*robot.result.executionerrors.ExecutionErrors attribute*), 412
messages (*robot.result.model.Keyword attribute*), 490
Metadata (*class in robot.model.metadata*), 264
Metadata (*class in robot.parsing.model.statements*), 354
metadata (*robot.model.testsuite.TestSuite attribute*), 285
METADATA (*robot.parsing.lexer.tokens.END attribute*), 339
METADATA (*robot.parsing.lexer.tokens.EOS attribute*), 337
METADATA (*robot.parsing.lexer.tokens.Token attribute*), 334
metadata (*robot.result.model.TestSuite attribute*), 497
metadata (*robot.running.model.TestSuite attribute*), 575
metadata_setting (*robot.conf.languages.Bg attribute*), 59
metadata_setting (*robot.conf.languages.Bs attribute*), 38
metadata_setting (*robot.conf.languages.Cs attribute*), 36
metadata_setting (*robot.conf.languages.De attribute*), 42
metadata_setting (*robot.conf.languages.En attribute*), 34
metadata_setting (*robot.conf.languages.Es attribute*), 51
metadata_setting (*robot.conf.languages.Fi attribute*), 40
metadata_setting (*robot.conf.languages.Fr attribute*), 41
metadata_setting (*robot.conf.languages.Hi attribute*), 63
metadata_setting (*robot.conf.languages.It attribute*), 62
metadata_setting (*robot.conf.languages.Language attribute*), 33
metadata_setting (*robot.conf.languages.Nl attribute*), 37
metadata_setting (*robot.conf.languages.Pl attribute*), 48
metadata_setting (*robot.conf.languages.Pt attribute*), 45
metadata_setting (*robot.conf.languages.PtBr attribute*), 44
metadata_setting (*robot.conf.languages.Ro attribute*), 61
metadata_setting (*robot.conf.languages.Ru attribute*), 52
metadata_setting (*robot.conf.languages.Sv attribute*), 58
metadata_setting (*robot.conf.languages.Th attribute*), 47
metadata_setting (*robot.conf.languages.Tr attribute*), 56
metadata_setting (*robot.conf.languages.Uk attribute*), 49
metadata_setting (*robot.conf.languages.ZhCn attribute*), 54
metadata_setting (*robot.conf.languages.ZhTw attribute*), 55
MetadataHandler (*class in robot.result.xmlelementhandlers*), 523
MetadataItemHandler (*class in robot.result.xmlelementhandlers*), 523
MetaHandler (*class in robot.result.xmlelementhandlers*), 524
minargs (*robot.running.arguments.argumentspec.ArgumentSpec attribute*), 530
minsize() (*robot.libraries.dialogs_py.InputDialog method*), 177
minsize() (*robot.libraries.dialogs_py.MessageDialog method*), 163
minsize() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 205
minsize() (*robot.libraries.dialogs_py.PassFailDialog method*), 219
minsize() (*robot.libraries.dialogs_py.SelectionDialog method*), 191
model_class (*robot.parsing.parser.fileparser.CommentSectionParser attribute*), 391
model_class (*robot.parsing.parser.fileparser.KeywordSectionParser attribute*), 391
model_class (*robot.parsing.parser.fileparser.SectionParser attribute*), 390
model_class (*robot.parsing.parser.fileparser.SettingSectionParser attribute*), 391
model_class (*robot.parsing.parser.fileparser.TestCaseSectionParser attribute*), 391

model_class (*robot.parsing.parser.fileparser.VariableSectionParser attribute*), 391
model_class() (*robot.parsing.parser.fileparser.ImplicitCommentSectionParser method*), 391
ModelCombiner (*class in robot.running.modelcombiner*), 577
ModelModifier (*class in robot.model.modifier*), 265
ModelObject (*class in robot.model.modelobject*), 265
ModelTransformer (*class in robot.parsing.model.visitor*), 389
ModelValidator (*class in robot.parsing.model.blocks*), 345
ModelVisitor (*class in robot.parsing.model.visitor*), 388
ModelWriter (*class in robot.htmldata.htmlfilewriter*), 67
ModelWriter (*class in robot.parsing.model.blocks*), 344
move_directory() (*robot.libraries.OperatingSystem.OperatingSystem method*), 118
move_file() (*robot.libraries.OperatingSystem.OperatingSystem method*), 118
move_files() (*robot.libraries.OperatingSystem.OperatingSystem method*), 118
move_to_end() (*robot.utils.dotdict.DotDict method*), 596
msg() (*robot.libraries.Telnet.TelnetConnection method*), 144
mt_interact() (*robot.libraries.Telnet.TelnetConnection method*), 145
multi_use (*robot.parsing.lexer.settings.InitFileSettings attribute*), 326
multi_use (*robot.parsing.lexer.settings.KeywordSettings attribute*), 327
multi_use (*robot.parsing.lexer.settings.ResourceFileSettings attribute*), 327
multi_use (*robot.parsing.lexer.settings.Settings attribute*), 326
multi_use (*robot.parsing.lexer.settings.TestCaseFileSettings attribute*), 326
multi_use (*robot.parsing.lexer.settings.TestCaseSettings attribute*), 327
MultiMatcher (*class in robot.utils.match*), 603
MultipleSelectionDialog (*class in robot.libraries.dialogs_py*), 199
MultiValue (*class in robot.parsing.model.statements*), 348

N

name (*robot.conf.languages.Bg attribute*), 60
name (*robot.conf.languages.Bs attribute*), 39
name (*robot.conf.languages.Cs attribute*), 37
name (*robot.conf.languages.De attribute*), 43
name (*robot.conf.languages.En attribute*), 35
name (*robot.conf.languages.Es attribute*), 52
name (*robot.conf.languages.Fi attribute*), 41
name (*robot.conf.languages.Fr attribute*), 42
name (*robot.conf.languages.Hi attribute*), 64
name (*robot.conf.languages.It attribute*), 63
name (*robot.conf.languages.Language attribute*), 34
name (*robot.conf.languages.Nl attribute*), 38
name (*robot.conf.languages.Pl attribute*), 49
name (*robot.conf.languages.Pt attribute*), 46
name (*robot.conf.languages.PtBr attribute*), 45
name (*robot.conf.languages.Ro attribute*), 62
name (*robot.conf.languages.Ru attribute*), 53
name (*robot.conf.languages.Sv attribute*), 59
name (*robot.conf.languages.Th attribute*), 48
name (*robot.conf.languages.Tr attribute*), 57
name (*robot.conf.languages.Uk attribute*), 50
name (*robot.conf.languages.ZhCn attribute*), 55
name (*robot.conf.languages.ZhTw attribute*), 56
name (*robot.model.keyword.Keyword attribute*), 259
name (*robot.model.stats.Stat attribute*), 274
name (*robot.model.testcase.TestCase attribute*), 283
name (*robot.model.testsuite.TestSuite attribute*), 285
name (*robot.output.pyloggingconf.RobotHandler attribute*), 313
NAME (*robot.parsing.lexer.tokens.END attribute*), 339
NAME (*robot.parsing.lexer.tokens.EOS attribute*), 337
NAME (*robot.parsing.lexer.tokens.Token attribute*), 334
name (*robot.parsing.model.blocks.Keyword attribute*), 343
name (*robot.parsing.model.blocks.TestCase attribute*), 343
name (*robot.parsing.model.statements.Fixture attribute*), 349
name (*robot.parsing.model.statements.KeywordName attribute*), 364
name (*robot.parsing.model.statements.LibraryImport attribute*), 351
name (*robot.parsing.model.statements.Metadata attribute*), 354
name (*robot.parsing.model.statements.ResourceImport attribute*), 351
name (*robot.parsing.model.statements.SectionHeader attribute*), 350
name (*robot.parsing.model.statements.Setup attribute*), 365
name (*robot.parsing.model.statements.SuiteSetup attribute*), 358
name (*robot.parsing.model.statements.SuiteTeardown attribute*), 359
name (*robot.parsing.model.statements.Teardown attribute*), 366
name (*robot.parsing.model.statements.TestCaseName attribute*), 363

name (*robot.parsing.model.statements.TestSetup attribute*), 359
name (*robot.parsing.model.statements.TestTeardown attribute*), 360
name (*robot.parsing.model.statements.Variable attribute*), 362
name (*robot.parsing.model.statements.VariablesImport attribute*), 352
name (*robot.result.model.Break attribute*), 489
name (*robot.result.model.Continue attribute*), 487
name (*robot.result.model.For attribute*), 467
name (*robot.result.model.ForIteration attribute*), 465
name (*robot.result.model.If attribute*), 478
name (*robot.result.model.IfBranch attribute*), 474
name (*robot.result.model.Keyword attribute*), 491
name (*robot.result.model.Return attribute*), 485
name (*robot.result.model.TestCase attribute*), 495
name (*robot.result.model.TestSuite attribute*), 497
name (*robot.result.model.Try attribute*), 482
name (*robot.result.model.TryBranch attribute*), 478
name (*robot.result.model.While attribute*), 472
name (*robot.result.model.WhileIteration attribute*), 469
name (*robot.result.modeldeprecation.DeprecatedAttributesMixin attribute*), 500
name (*robot.running.arguments.customconverters.ConverterInfo attribute*), 531
name (*robot.running.dynamicmethods.GetKeywordArguments attribute*), 551
name (*robot.running.dynamicmethods.GetKeywordDocumentation attribute*), 551
name (*robot.running.dynamicmethods.GetKeywordNames attribute*), 550
name (*robot.running.dynamicmethods.GetKeywordSource attribute*), 551
name (*robot.running.dynamicmethods.GetKeywordTags attribute*), 551
name (*robot.running.dynamicmethods.GetKeywordTypes attribute*), 551
name (*robot.running.dynamicmethods.RunKeyword attribute*), 551
name (*robot.running.model.Keyword attribute*), 557
name (*robot.running.model.TestCase attribute*), 571
name (*robot.running.model.TestSuite attribute*), 575
name (*robot.variables.search.VariableMatch attribute*), 611
name_and_arguments (*robot.parsing.lexer.settings.InitFileSettings attribute*), 326
name_and_arguments (*robot.parsing.lexer.settings.KeywordSettings attribute*), 327
name_and_arguments (*robot.parsing.lexer.settings.ResourceFileSettings attribute*), 327
name_and_arguments (*robot.parsing.lexer.settings.Settings attribute*), 326
name_and_arguments (*robot.parsing.lexer.settings.TestCaseFileSettings attribute*), 326
name_and_arguments (*robot.parsing.lexer.settings.TestCaseSettings attribute*), 327
name_arguments_and_with_name (*robot.parsing.lexer.settings.InitFileSettings attribute*), 326
name_arguments_and_with_name (*robot.parsing.lexer.settings.KeywordSettings attribute*), 327
name_arguments_and_with_name (*robot.parsing.lexer.settings.ResourceFileSettings attribute*), 327
name_arguments_and_with_name (*robot.parsing.lexer.settings.Settings attribute*), 326
name_arguments_and_with_name (*robot.parsing.lexer.settings.TestCaseFileSettings attribute*), 326
name_arguments_and_with_name (*robot.parsing.lexer.settings.TestCaseSettings attribute*), 327
name_type (*robot.parsing.lexer.blocklexers.KeywordLexer attribute*), 321
name_type (*robot.parsing.lexer.blocklexers.TestCaseLexer attribute*), 321
name_type (*robot.parsing.lexer.blocklexers.TestOrKeywordLexer attribute*), 321
NAMED_ONLY (*robot.running.arguments.argumentspec.ArgInfo attribute*), 530
NAMED_ONLY_MARKER (*robot.running.arguments.argumentspec.ArgInfo attribute*), 530
NamedArgumentResolver (*class in robot.running.arguments.argumentresolver*), 529
NamePatterns (*class in robot.model.namepatterns*), 270
names (*robot.parsing.lexer.settings.InitFileSettings attribute*), 326
names (*robot.parsing.lexer.settings.KeywordSettings attribute*), 327
names (*robot.parsing.lexer.settings.ResourceFileSettings attribute*), 326
names (*robot.parsing.lexer.settings.Settings attribute*), 326
names (*robot.parsing.lexer.settings.TestCaseFileSettings attribute*), 326
names (*robot.parsing.lexer.settings.TestCaseSettings at-*

tribute), 327
Namespace (*class in robot.running.namespace*), 577
namespaces (*robot.running.context.ExecutionContexts attribute*), 550
NameSpaceStripper (*class in robot.libraries.XML*), 157
nametowidget() (*robot.libraries.dialogs_py.InputDialog method*), 177
nametowidget() (*robot.libraries.dialogs_py.MessageDialog method*), 163
nametowidget() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 205
nametowidget() (*robot.libraries.dialogs_py.PassFailDialog method*), 219
nametowidget() (*robot.libraries.dialogs_py.SelectionDialog method*), 191
NestedBlockLexer (*class in robot.parsing.lexer.blocklexers*), 322
NestedBlockParser (*class in robot.parsing.parser.blockparsers*), 390
NEW_ENVIRON_IS (*robot.libraries.Telnet.TelnetConnection attribute*), 141
NEW_ENVIRON_VALUE (*robot.libraries.Telnet.TelnetConnection attribute*), 141
NEW_ENVIRON_VAR (*robot.libraries.Telnet.TelnetConnection attribute*), 141
new_suite_scope() (*robot.output.listenermethods.LibraryListenerMethods method*), 307
new_suite_scope() (*robot.output.listeners.LibraryListeners method*), 307
newline (*robot.utils.htmlformatters.LineFormatter attribute*), 599
Nl (*class in robot.conf.languages*), 37
no_conversion_needed() (*robot.running.arguments.typeconverters.BooleanConverter method*), 533
no_conversion_needed() (*robot.running.arguments.typeconverters.ByteArrayConverter method*), 535
no_conversion_needed() (*robot.running.arguments.typeconverters.BytesConverter method*), 535
no_conversion_needed() (*robot.running.arguments.typeconverters.CombinedConverter method*), 540
no_conversion_needed() (*robot.running.arguments.typeconverters.CustomConverter method*), 541
no_conversion_needed() (*robot.running.arguments.typeconverters.DateConverter method*), 536
no_conversion_needed() (*robot.running.arguments.typeconverters.DateTimeConverter method*), 536
no_conversion_needed() (*robot.running.arguments.typeconverters.DecimalConverter method*), 534
no_conversion_needed() (*robot.running.arguments.typeconverters.DictionaryConverter method*), 539
no_conversion_needed() (*robot.running.arguments.typeconverters.EnumConverter method*), 532
no_conversion_needed() (*robot.running.arguments.typeconverters.FloatConverter method*), 534
no_conversion_needed() (*robot.running.arguments.typeconverters.FrozenSetConverter method*), 540
no_conversion_needed() (*robot.running.arguments.typeconverters.IntegerConverter method*), 533
no_conversion_needed() (*robot.running.arguments.typeconverters.ListConverter method*), 538
no_conversion_needed() (*robot.running.arguments.typeconverters.NoneConverter method*), 537
no_conversion_needed() (*robot.running.arguments.typeconverters.PathConverter method*), 537
no_conversion_needed() (*robot.running.arguments.typeconverters.SetConverter method*), 539
no_conversion_needed() (*robot.running.arguments.typeconverters.StringConverter method*), 533
no_conversion_needed() (*robot.running.arguments.typeconverters.TimeDeltaConverter method*), 536
no_conversion_needed() (*robot.running.arguments.typeconverters.TupleConverter method*), 538
no_conversion_needed() (*robot.running.arguments.typeconverters.TypeConverter method*), 532
no_conversion_needed() (*robot.running.arguments.typeconverters.TypedDictConverter method*), 538
no_dynamic_method() (*in module robot.running.dynamicmethods*), 550
no_operation() (*robot.libraries.BuiltIn.BuiltIn method*), 84
NoArgumentHeader (*class in robot.parsing.model.statements*), 377

NoConnection (*class in robot.utils.connectioncache*), 596
NoHighlighting (*class in robot.output.console.highlighting*), 303
NoInitFileDirectoryParser (*class in robot.running.builder.parsers*), 543
NoLimit (*class in robot.running.bodyrunner*), 550
NoLogger (*class in robot.utils.importer*), 602
NoMatchError, 146
non_ascii (*robot.libraries.Remote.ArgumentCoercer attribute*), 129
NON_DATA_TOKENS (*robot.parsing.lexer.tokens.END attribute*), 339
NON_DATA_TOKENS (*robot.parsing.lexer.tokens.EOS attribute*), 337
NON_DATA_TOKENS (*robot.parsing.lexer.tokens.Token attribute*), 335
NonDottedImporter (*class in robot.utils.importer*), 601
none_shall_pass() (*in module robot.libraries.Easter*), 112
NoneConverter (*class in robot.running.arguments.typeconverters*), 537
NoneDumper (*class in robot.htmldata.jsonwriter*), 68
NoOutput (*class in robot.output.console.quiet*), 304
NoReturnValueResolver (*class in robot.variables.assigner*), 606
normal (*robot.model.keyword.Keywords attribute*), 261
normalize() (*in module robot.utils.normalizing*), 604
normalize_path() (*robot.libraries.OperatingSystem.OperatingSystem method*), 120
normalize_whitespace() (*in module robot.utils.normalizing*), 604
NormalizedDict (*class in robot.utils.normalizing*), 604
not_keyword() (*in module robot.api.deco*), 11
NOT_RUN (*robot.result.model.Break attribute*), 488
not_run (*robot.result.model.Break attribute*), 489
NOT_RUN (*robot.result.model.Continue attribute*), 486
not_run (*robot.result.model.Continue attribute*), 487
NOT_RUN (*robot.result.model.For attribute*), 467
not_run (*robot.result.model.For attribute*), 469
NOT_RUN (*robot.result.model.ForIteration attribute*), 465
not_run (*robot.result.model.ForIteration attribute*), 466
NOT_RUN (*robot.result.model.If attribute*), 477
not_run (*robot.result.model.If attribute*), 478
NOT_RUN (*robot.result.model.IfBranch attribute*), 474
not_run (*robot.result.model.IfBranch attribute*), 476
NOT_RUN (*robot.result.model.Keyword attribute*), 491
not_run (*robot.result.model.Keyword attribute*), 492
NOT_RUN (*robot.result.model.Return attribute*), 484
not_run (*robot.result.model.Return attribute*), 485
NOT_RUN (*robot.result.model.StatusMixin attribute*), 464
not_run (*robot.result.model.StatusMixin attribute*), 464
NOT_RUN (*robot.result.model.TestCase attribute*), 493
not_run (*robot.result.model.TestCase attribute*), 493
NOT_RUN (*robot.result.model.TestSuite attribute*), 496
not_run (*robot.result.model.TestSuite attribute*), 497
NOT_RUN (*robot.result.model.Try attribute*), 481
not_run (*robot.result.model.Try attribute*), 482
NOT_RUN (*robot.result.model.TryBranch attribute*), 479
not_run (*robot.result.model.TryBranch attribute*), 480
NOT_RUN (*robot.result.model.While attribute*), 472
not_run (*robot.result.model.While attribute*), 473
NOT_RUN (*robot.result.model.WhileIteration attribute*), 470
not_run (*robot.result.model.WhileIteration attribute*), 471
NOT_SET (*robot.result.model.Break attribute*), 488
NOT_SET (*robot.result.model.Continue attribute*), 486
NOT_SET (*robot.result.model.For attribute*), 467
NOT_SET (*robot.result.model.ForIteration attribute*), 465
NOT_SET (*robot.result.model.If attribute*), 477
NOT_SET (*robot.result.model.IfBranch attribute*), 474
NOT_SET (*robot.result.model.Keyword attribute*), 491
NOT_SET (*robot.result.model.Return attribute*), 484
NOT_SET (*robot.result.model.StatusMixin attribute*), 464
NOT_SET (*robot.result.model.TestCase attribute*), 493
NOT_SET (*robot.result.model.TestSuite attribute*), 496
NOT_SET (*robot.result.model.Try attribute*), 481
NOT_SET (*robot.result.model.TryBranch attribute*), 479
NOT_SET (*robot.result.model.While attribute*), 472
NOT_SET (*robot.result.model.WhileIteration attribute*), 470
NotSet (*class in robot.libraries.Collections*), 100
NOTSET (*robot.running.arguments.argumentspec.ArgInfo attribute*), 530
NotTagPattern (*class in robot.model.tags*), 277
NullMarkupWriter (*class in robot.utils.markupwriters*), 603
NullNamedArgumentResolver (*class in robot.running.arguments.argumentresolver*), 529
NumberFinder (*class in robot.variables.finders*), 608
numerator (*robot.reporting.stringcache.StringIndex attribute*), 401

O

OneReturnValueResolver (*class in robot.variables.assigner*), 607
open() (*robot.libraries.Telnet.TelnetConnection method*), 145
open_connection() (*robot.libraries.Telnet.Telnet method*), 141

OperatingSystem (*class in robot.libraries.OperatingSystem*), 112
OPTION (*robot.parsing.lexer.tokens.END attribute*), 339
OPTION (*robot.parsing.lexer.tokens.EOS attribute*), 337
OPTION (*robot.parsing.lexer.tokens.Token attribute*), 335
option_add() (*robot.libraries.dialogs_py.InputDialog method*), 177
option_add() (*robot.libraries.dialogs_py.MessageDialog method*), 163
option_add() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 205
option_add() (*robot.libraries.dialogs_py.PassFailDialog method*), 219
option_add() (*robot.libraries.dialogs_py.SelectionDialog method*), 191
option_clear() (*robot.libraries.dialogs_py.InputDialog method*), 178
option_clear() (*robot.libraries.dialogs_py.MessageDialog method*), 164
option_clear() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 206
option_clear() (*robot.libraries.dialogs_py.PassFailDialog method*), 220
option_clear() (*robot.libraries.dialogs_py.SelectionDialog method*), 192
option_get() (*robot.libraries.dialogs_py.InputDialog method*), 178
option_get() (*robot.libraries.dialogs_py.MessageDialog method*), 164
option_get() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 206
option_get() (*robot.libraries.dialogs_py.PassFailDialog method*), 220
option_get() (*robot.libraries.dialogs_py.SelectionDialog method*), 192
option_readfile() (*robot.libraries.dialogs_py.InputDialog method*), 178
option_readfile() (*robot.libraries.dialogs_py.MessageDialog method*), 164
option_readfile() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 206
option_readfile() (*robot.libraries.dialogs_py.PassFailDialog method*), 220
option_readfile() (*robot.libraries.dialogs_py.SelectionDialog method*), 192
OrTagPattern (*class in robot.model.tags*), 277
Output (*class in robot.output.output*), 311
output (*robot.conf.settings.RebotSettings attribute*), 66
output (*robot.conf.settings.RobotSettings attribute*), 66
output() (*robot.output.console.verbose.VerboseWriter method*), 304
output_directory (*robot.conf.settings.RebotSettings attribute*), 66
output_directory (*robot.conf.settings.RobotSettings attribute*), 66
output_file() (*robot.output.console.dotted.DottedOutput method*), 299
output_file() (*robot.output.console.verbose.VerboseOutput method*), 304
output_file() (*robot.output.filelogger.FileLogger method*), 305
output_file() (*robot.output.listeners.LibraryListeners method*), 307
output_file() (*robot.output.listeners.Listeners method*), 307
output_file() (*robot.output.logger.Logger method*), 308
OutputCapturer (*class in robot.running.outputcapture*), 578
OutputWriter (*class in robot.reporting.outputwriter*), 395
overrideredirect() (*robot.libraries.dialogs_py.InputDialog method*), 178
overrideredirect() (*robot.libraries.dialogs_py.MessageDialog method*), 164
overrideredirect() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 206
overrideredirect() (*robot.libraries.dialogs_py.PassFailDialog method*), 220
overrideredirect() (*robot.libraries.dialogs_py.SelectionDialog method*), 192

P

pack_propagate() (*robot.libraries.dialogs_py.InputDialog method*), 178
pack_propagate() (*robot.libraries.dialogs_py.MessageDialog method*), 164
pack_propagate() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 206
pack_propagate() (*robot.libraries.dialogs_py.PassFailDialog method*), 220
pack_propagate() (*robot.libraries.dialogs_py.SelectionDialog method*), 192
pack_slaves() (*robot.libraries.dialogs_py.InputDialog method*), 178
pack_slaves() (*robot.libraries.dialogs_py.MessageDialog method*), 164

pack_slaves() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 206
pack_slaves() (*robot.libraries.dialogs_py.PassFailDialog method*), 220
pack_slaves() (*robot.libraries.dialogs_py.SelectionDialog method*), 192
ParagraphFormatter (*class in robot.utils.htmlformatters*), 600
parent (*robot.model.body.BodyItem attribute*), 229
parent (*robot.model.control.Break attribute*), 250
parent (*robot.model.control.Continue attribute*), 248
parent (*robot.model.control.For attribute*), 239
parent (*robot.model.control.If attribute*), 242
parent (*robot.model.control.IfBranch attribute*), 241
parent (*robot.model.control.Return attribute*), 246
parent (*robot.model.control.Try attribute*), 245
parent (*robot.model.control.TryBranch attribute*), 244
parent (*robot.model.control.While attribute*), 240
parent (*robot.model.keyword.Keyword attribute*), 259
parent (*robot.model.message.Message attribute*), 262
parent (*robot.model.testcase.TestCase attribute*), 283
parent (*robot.model.testsuite.TestSuite attribute*), 285
parent (*robot.output.loggerhelper.Message attribute*), 311
parent (*robot.result.model.Break attribute*), 489
parent (*robot.result.model.Continue attribute*), 487
parent (*robot.result.model.For attribute*), 469
parent (*robot.result.model.ForIteration attribute*), 465
parent (*robot.result.model.If attribute*), 478
parent (*robot.result.model.IfBranch attribute*), 476
parent (*robot.result.model.Keyword attribute*), 492
parent (*robot.result.model.Message attribute*), 464
parent (*robot.result.model.Return attribute*), 485
parent (*robot.result.model.TestCase attribute*), 495
parent (*robot.result.model.TestSuite attribute*), 497
parent (*robot.result.model.Try attribute*), 482
parent (*robot.result.model.TryBranch attribute*), 480
parent (*robot.result.model.While attribute*), 473
parent (*robot.result.model.WhileIteration attribute*), 469
parent (*robot.running.model.Break attribute*), 570
parent (*robot.running.model.Continue attribute*), 568
parent (*robot.running.model.For attribute*), 559
parent (*robot.running.model.If attribute*), 563
parent (*robot.running.model.IfBranch attribute*), 561
parent (*robot.running.model.Keyword attribute*), 557
parent (*robot.running.model.Return attribute*), 567
parent (*robot.running.model.TestCase attribute*), 571
parent (*robot.running.model.TestSuite attribute*), 575
parent (*robot.running.model.Try attribute*), 566
parent (*robot.running.model.TryBranch attribute*), 564
parent (*robot.running.model.While attribute*), 560
ParentMessage (*class in robot.running.status*), 584
parse() (*robot.parsing.parser.blockparsers.BlockParser method*), 389
parse() (*robot.parsing.parser.blockparsers.ForParser method*), 390
parse() (*robot.parsing.parser.blockparsers.IfParser method*), 390
parse() (*robot.parsing.parser.blockparsers.KeywordParser method*), 389
parse() (*robot.parsing.parser.blockparsers.NestedBlockParser method*), 390
parse() (*robot.parsing.parser.blockparsers.Parser method*), 389
parse() (*robot.parsing.parser.blockparsers.TestCaseParser method*), 389
parse() (*robot.parsing.parser.blockparsers.TryParser method*), 390
parse() (*robot.parsing.parser.blockparsers.WhileParser method*), 390
parse() (*robot.parsing.parser.fileparser.CommentSectionParser method*), 391
parse() (*robot.parsing.parser.fileparser.FileParser method*), 390
parse() (*robot.parsing.parser.fileparser.ImplicitCommentSectionParser method*), 391
parse() (*robot.parsing.parser.fileparser.KeywordSectionParser method*), 391
parse() (*robot.parsing.parser.fileparser.SectionParser method*), 390
parse() (*robot.parsing.parser.fileparser.SettingSectionParser method*), 391
parse() (*robot.parsing.parser.fileparser.TestCaseSectionParser method*), 391
parse() (*robot.parsing.parser.fileparser.VariableSectionParser method*), 391
parse() (*robot.running.arguments.argumentparser.DynamicArgumentPar method*), 529
parse() (*robot.running.arguments.argumentparser.PythonArgumentParse method*), 529
parse() (*robot.running.arguments.argumentparser.UserKeywordArgumer method*), 529
parse() (*robot.running.arguments.embedded.EmbeddedArgumentParser method*), 531
parse() (*robot.running.builder.builders.SuiteStructureParser method*), 542
parse_args() (*robot.utils.argumentparser.ArgumentParser method*), 592
parse_arguments() (*robot.libdoc.LibDoc method*), 619
parse_arguments() (*robot.rebot.Rebot method*), 620
parse_arguments() (*robot.run.RobotFramework method*), 622
parse_arguments() (*robot.testdoc.TestDoc method*), 624

parse_arguments() (*robot.utils.application.Application method*), 591
parse_init_file() (*robot.running.builder.parsers.BaseParser method*), 542
parse_init_file() (*robot.running.builder.parsers.NoInitFileDirectoryParser method*), 543
parse_init_file() (*robot.running.builder.parsers.RestParser method*), 543
parse_init_file() (*robot.running.builder.parsers.RobotParser method*), 543
parse_re_flags() (*in module robot.utils.misc*), 604
parse_resource_file() (*robot.running.builder.parsers.BaseParser method*), 542
parse_resource_file() (*robot.running.builder.parsers.NoInitFileDirectoryParser method*), 543
parse_resource_file() (*robot.running.builder.parsers.RestParser method*), 543
parse_resource_file() (*robot.running.builder.parsers.RobotParser method*), 543
parse_response() (*robot.libraries.Remote.TimeoutHTTPTransport method*), 130
parse_response() (*robot.libraries.Remote.TimeoutHTTPTransport method*), 129
parse_suite_file() (*robot.running.builder.parsers.BaseParser method*), 542
parse_suite_file() (*robot.running.builder.parsers.NoInitFileDirectoryParser method*), 543
parse_suite_file() (*robot.running.builder.parsers.RestParser method*), 543
parse_suite_file() (*robot.running.builder.parsers.RobotParser method*), 543
parse_xml() (*robot.libraries.XML.XML method*), 150
Parser (*class in robot.parsing.parser.blockparsers*), 389
PASS (*robot.result.model.Break attribute*), 488
PASS (*robot.result.model.Continue attribute*), 486
PASS (*robot.result.model.For attribute*), 467
PASS (*robot.result.model.ForIteration attribute*), 465
PASS (*robot.result.model.If attribute*), 477
PASS (*robot.result.model.IfBranch attribute*), 474
PASS (*robot.result.model.Keyword attribute*), 491
PASS (*robot.result.model.Return attribute*), 484
PASS (*robot.result.model.StatusMixin attribute*), 464
PASS (*robot.result.model.TestCase attribute*), 494
PASS (*robot.result.model.TestSuite attribute*), 496
PASS (*robot.result.model.Try attribute*), 481
PASS (*robot.result.model.TryBranch attribute*), 479
PASS (*robot.result.model.While attribute*), 472
PASS (*robot.result.model.WhileIteration attribute*), 470
pass_execution() (*robot.libraries.BuiltIn.BuiltIn method*), 84
pass_execution_if() (*robot.libraries.BuiltIn.BuiltIn method*), 84
passed (*robot.model.stats.Stat attribute*), 275
passed (*robot.model.totalstatistics.TotalStatistics attribute*), 288
passed (*robot.result.model.Break attribute*), 489
passed (*robot.result.model.Continue attribute*), 487
passed (*robot.result.model.For attribute*), 469
passed (*robot.result.model.ForIteration attribute*), 466
passed (*robot.result.model.If attribute*), 478
passed (*robot.result.model.IfBranch attribute*), 476
passed (*robot.result.model.Keyword attribute*), 492
passed (*robot.result.model.Return attribute*), 485
passed (*robot.result.model.StatusMixin attribute*), 464
passed (*robot.result.model.TestCase attribute*), 495
passed (*robot.result.model.TestSuite attribute*), 496
passed (*robot.result.model.Try attribute*), 482
passed (*robot.result.model.TryBranch attribute*), 480
passed (*robot.result.model.While attribute*), 473
passed (*robot.result.model.WhileIteration attribute*), 471
passed (*robot.running.status.SuiteStatus attribute*), 583
passed (*robot.running.status.TestStatus attribute*), 583
PassedKeywordRemover (*class in robot.result.keywordremover*), 419
PassExecution, 616
PassFailDialog (*class in robot.libraries.dialogs_py*), 213
PathConverter (*class in robot.running.arguments.typeconverters*), 536
pattern_type (*robot.model.control.TryBranch attribute*), 244
pattern_type (*robot.parsing.model.blocks.Try attribute*), 344
pattern_type (*robot.parsing.model.statements.ExceptHeader attribute*), 379
pattern_type (*robot.result.model.TryBranch attribute*), 480
pattern_type (*robot.running.model.TryBranch attribute*), 564
PatternHandler (*class in robot.result.xmlelementhandlers*), 522

patterns (*robot.model.control.TryBranch attribute*), 244
patterns (*robot.parsing.model.blocks.Try attribute*), 344
patterns (*robot.parsing.model.statements.ExceptHeader attribute*), 379
patterns (*robot.result.model.TryBranch attribute*), 480
patterns (*robot.running.model.TryBranch attribute*), 564
pause_execution() (*in module robot.libraries.Dialogs*), 111
Pl (*class in robot.conf.languages*), 48
place_slaves() (*robot.libraries.dialogs_py.InputDialog method*), 178
place_slaves() (*robot.libraries.dialogs_py.MessageDialog method*), 164
place_slaves() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 206
place_slaves() (*robot.libraries.dialogs_py.PassFailDialog method*), 220
place_slaves() (*robot.libraries.dialogs_py.SelectionDialog method*), 192
plural_or_not() (*in module robot.utils.misc*), 604
pop() (*robot.model.body.BaseBody method*), 230
pop() (*robot.model.body.Body method*), 232
pop() (*robot.model.body.Branches method*), 233
pop() (*robot.model.itemlist.ItemList method*), 259
pop() (*robot.model.keyword.Keywords method*), 262
pop() (*robot.model.message.Messages method*), 264
pop() (*robot.model.metadata.Metadata method*), 264
pop() (*robot.model.testcase.TestCases method*), 285
pop() (*robot.model.testsuite.TestSuites method*), 288
pop() (*robot.result.model.Body method*), 459
pop() (*robot.result.model.Branches method*), 461
pop() (*robot.result.model.Iterations method*), 462
pop() (*robot.running.model.Body method*), 555
pop() (*robot.running.model.Imports method*), 577
pop() (*robot.utils.dotdict.DotDict method*), 597
pop() (*robot.utils.normalizing.NormalizedDict method*), 605
pop() (*robot.variables.evaluation.EvaluationNamespace method*), 607
pop_from_dictionary() (*robot.libraries.Collections.Collections method*), 105
popen_config (*robot.libraries.Process.ProcessConfiguration attribute*), 128
popitem() (*robot.model.metadata.Metadata method*), 264
popitem() (*robot.utils.dotdict.DotDict method*), 597
popitem() (*robot.utils.normalizing.NormalizedDict method*), 605
popitem() (*robot.variables.evaluation.EvaluationNamespace method*), 607
positional (*robot.running.arguments.argumentspec.ArgumentSpec attribute*), 530
POSITIONAL_ONLY (*robot.running.arguments.argumentspec.ArgInfo attribute*), 530
POSITIONAL_ONLY_MARKER (*robot.running.arguments.argumentspec.ArgInfo attribute*), 530
POSITIONAL_OR_NAMED (*robot.running.arguments.argumentspec.ArgInfo attribute*), 530
positionfrom() (*robot.libraries.dialogs_py.InputDialog method*), 178
positionfrom() (*robot.libraries.dialogs_py.MessageDialog method*), 164
positionfrom() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 206
positionfrom() (*robot.libraries.dialogs_py.PassFailDialog method*), 220
positionfrom() (*robot.libraries.dialogs_py.SelectionDialog method*), 192
post_rebot_modifiers (*robot.conf.settings.RebotSettings attribute*), 66
pre_rebot_modifiers (*robot.conf.settings.RobotSettings attribute*), 66
pre_run_modifiers (*robot.conf.settings.RobotSettings attribute*), 65
PreformattedFormatter (*class in robot.utils.htmlformatters*), 600
printable_name() (*in module robot.utils.misc*), 603
priority (*robot.running.modelcombiner.ModelCombiner attribute*), 577
private (*robot.running.userkeyword.EmbeddedArgumentsHandler attribute*), 590
private (*robot.running.userkeyword.UserKeywordHandler attribute*), 590
Process (*class in robot.libraries.Process*), 122
process() (*robot.utils.argumentparser.ArgFileParser method*), 592
process_empty_suite (*robot.conf.settings.RebotSettings attribute*), 66
process_rawq() (*robot.libraries.Telnet.TelnetConnection method*), 145
process_should_be_running() (*robot.libraries.Process.Process method*), 125
process_should_be_stopped() (*robot.libraries.Process.Process method*), 126
ProcessConfiguration (*class in robot.libraries.Process*), 128
propagate() (*robot.libraries.dialogs_py.InputDialog method*), 178
propagate() (*robot.libraries.dialogs_py.MessageDialog*

method), 164
propagate() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 206
propagate() (*robot.libraries.dialogs_py.PassFailDialog method*), 220
propagate() (*robot.libraries.dialogs_py.SelectionDialog method*), 192
protocol() (*robot.libraries.dialogs_py.InputDialog method*), 178
protocol() (*robot.libraries.dialogs_py.MessageDialog method*), 164
protocol() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 206
protocol() (*robot.libraries.dialogs_py.PassFailDialog method*), 220
protocol() (*robot.libraries.dialogs_py.SelectionDialog method*), 192
prune_input() (*robot.reporting.jsbuildingcontext.JsBuildingContext method*), 394
Pt (*class in robot.conf.languages*), 45
PtBr (*class in robot.conf.languages*), 44
py2to3() (*in module robot.utils*), 591
py3to2() (*in module robot.utils*), 591
PythonArgumentParser (*class in robot.running.arguments.argumentparser*), 529
PythonCapturer (*class in robot.running.outputcapture*), 578
PythonImporter (*class in robot.variables.filesetter*), 607
pythonpath (*robot.conf.settings.RebotSettings attribute*), 66
pythonpath (*robot.conf.settings.RobotSettings attribute*), 66

Q

QuietOutput (*class in robot.output.console.quiet*), 304
quit() (*robot.libraries.dialogs_py.InputDialog method*), 178
quit() (*robot.libraries.dialogs_py.MessageDialog method*), 164
quit() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 206
quit() (*robot.libraries.dialogs_py.PassFailDialog method*), 220
quit() (*robot.libraries.dialogs_py.SelectionDialog method*), 192

R

raise_deprecation_error() (*robot.model.keyword.Keywords class method*), 262
raise_error() (*robot.utils.connectioncache.NoConnection method*), 596
randomize() (*robot.running.model.TestSuite method*), 573
randomize_seed (*robot.conf.settings.RobotSettings attribute*), 65
randomize_suites (*robot.conf.settings.RobotSettings attribute*), 65
randomize_tests (*robot.conf.settings.RobotSettings attribute*), 65
Randomizer (*class in robot.running.randomizer*), 578
read() (*robot.libraries.Telnet.TelnetConnection method*), 144
read() (*robot.libraries.Telnet.TerminalEmulator method*), 146
read() (*robot.utils.filereader.FileReader method*), 599
read_all() (*robot.libraries.Telnet.TelnetConnection method*), 145
read_eager() (*robot.libraries.Telnet.TelnetConnection method*), 145
read_lazy() (*robot.libraries.Telnet.TelnetConnection method*), 146
read_rest_data() (*in module robot.utils*), 591
read_sb_data() (*robot.libraries.Telnet.TelnetConnection method*), 146
read_some() (*robot.libraries.Telnet.TelnetConnection method*), 146
read_until() (*robot.libraries.Telnet.TelnetConnection method*), 144
read_until() (*robot.libraries.Telnet.TerminalEmulator method*), 146
read_until_prompt() (*robot.libraries.Telnet.TelnetConnection method*), 144
read_until_regexp() (*robot.libraries.Telnet.TelnetConnection method*), 144
read_until_regexp() (*robot.libraries.Telnet.TerminalEmulator method*), 146
read_very_eager() (*robot.libraries.Telnet.TelnetConnection method*), 146
read_very_lazy() (*robot.libraries.Telnet.TelnetConnection method*), 146
readlines() (*robot.utils.filereader.FileReader method*), 599
real (*robot.reporting.stringcache.StringIndex attribute*), 402
Rebot (*class in robot.rebot*), 620
rebot() (*in module robot*), 9
rebot() (*in module robot.rebot*), 621

rebot_cli() (*in module robot*), 9
rebot_cli() (*in module robot.rebot*), 621
RebotSettings (*class in robot.conf.settings*), 66
recommend_similar_keywords() (*robot.running.namespace.KeywordRecommendationFinder method*), 578
RecommendationFinder (*class in robot.utils.recommendations*), 605
red() (*robot.output.console.highlighting.AnsiHighlighter method*), 303
red() (*robot.output.console.highlighting.DosHighlighter method*), 303
red() (*robot.output.console.highlighting.NoHighlighting method*), 303
regexp_escape() (*robot.libraries.BuiltIn.BuiltIn method*), 84
register() (*robot.libraries.dialogs_py.InputDialog method*), 178
register() (*robot.libraries.dialogs_py.MessageDialog method*), 164
register() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 206
register() (*robot.libraries.dialogs_py.PassFailDialog method*), 220
register() (*robot.libraries.dialogs_py.SelectionDialog method*), 192
register() (*robot.model.body.BaseBody class method*), 229
register() (*robot.model.body.Body class method*), 232
register() (*robot.model.body.Branches class method*), 233
register() (*robot.output.listenermethods.LibraryListenerMethods method*), 307
register() (*robot.output.listeners.LibraryListeners method*), 307
register() (*robot.parsing.model.statements.Arguments class method*), 369
register() (*robot.parsing.model.statements.Break class method*), 385
register() (*robot.parsing.model.statements.Comment class method*), 386
register() (*robot.parsing.model.statements.Config class method*), 387
register() (*robot.parsing.model.statements.Continue class method*), 384
register() (*robot.parsing.model.statements.DefaultTags class method*), 356
register() (*robot.parsing.model.statements.Documentation class method*), 354
register() (*robot.parsing.model.statements.DocumentationOrMetadata class method*), 347
register() (*robot.parsing.model.statements.ElseHeader class method*), 377
register() (*robot.parsing.model.statements.ElseIfHeader class method*), 376
register() (*robot.parsing.model.statements.EmptyLine class method*), 388
register() (*robot.parsing.model.statements.End class method*), 381
register() (*robot.parsing.model.statements.Error class method*), 387
register() (*robot.parsing.model.statements.ExceptHeader class method*), 379
register() (*robot.parsing.model.statements.FinallyHeader class method*), 380
register() (*robot.parsing.model.statements.Fixture class method*), 349
register() (*robot.parsing.model.statements.ForceTags class method*), 355
register() (*robot.parsing.model.statements.ForHeader class method*), 372
register() (*robot.parsing.model.statements.IfElseHeader class method*), 373
register() (*robot.parsing.model.statements.IfHeader class method*), 374
register() (*robot.parsing.model.statements.InlineIfHeader class method*), 375
register() (*robot.parsing.model.statements.KeywordCall class method*), 371
register() (*robot.parsing.model.statements.KeywordName class method*), 364
register() (*robot.parsing.model.statements.KeywordTags class method*), 357
register() (*robot.parsing.model.statements.LibraryImport class method*), 351
register() (*robot.parsing.model.statements.LoopControl class method*), 383
register() (*robot.parsing.model.statements.Metadata class method*), 354
register() (*robot.parsing.model.statements.MultiValue class method*), 349
register() (*robot.parsing.model.statements.NoArgumentHeader class method*), 377
register() (*robot.parsing.model.statements.ResourceImport class method*), 352
register() (*robot.parsing.model.statements.Return class method*), 370
register() (*robot.parsing.model.statements.ReturnStatement class method*), 382
register() (*robot.parsing.model.statements.SectionHeader class method*), 350
register() (*robot.parsing.model.statements.Setup class method*), 365
register() (*robot.parsing.model.statements.SingleValue class method*), 348
register() (*robot.parsing.model.statements.Statement class method*), 346

register() (*robot.parsing.model.statements.SuiteSetup class method*), 358
register() (*robot.parsing.model.statements.SuiteTeardown class method*), 359
register() (*robot.parsing.model.statements.Tags class method*), 367
register() (*robot.parsing.model.statements.Teardown class method*), 366
register() (*robot.parsing.model.statements.Template class method*), 367
register() (*robot.parsing.model.statements.TemplateArguments class method*), 372
register() (*robot.parsing.model.statements.TestCaseName class method*), 363
register() (*robot.parsing.model.statements.TestSetup class method*), 359
register() (*robot.parsing.model.statements.TestTeardown class method*), 360
register() (*robot.parsing.model.statements.TestTemplate class method*), 361
register() (*robot.parsing.model.statements.TestTimeout class method*), 362
register() (*robot.parsing.model.statements.Timeout class method*), 368
register() (*robot.parsing.model.statements.TryHeader class method*), 378
register() (*robot.parsing.model.statements.Variable class method*), 363
register() (*robot.parsing.model.statements.VariablesImport class method*), 353
register() (*robot.parsing.model.statements.WhileHeader class method*), 382
register() (*robot.result.model.Body class method*), 459
register() (*robot.result.model.Branches class method*), 461
register() (*robot.result.model.Iterations class method*), 462
register() (*robot.result.xmlelementhandlers.ArgumentHandler class method*), 526
register() (*robot.result.xmlelementhandlers.ArgumentsHandler class method*), 525
register() (*robot.result.xmlelementhandlers.AssignHandler class method*), 525
register() (*robot.result.xmlelementhandlers.BranchHandler class method*), 521
register() (*robot.result.xmlelementhandlers.BreakHandler class method*), 522
register() (*robot.result.xmlelementhandlers.ContinueHandler class method*), 522
register() (*robot.result.xmlelementhandlers.DocHandler class method*), 523
register() (*robot.result.xmlelementhandlers.ElementHandler class method*), 519
register() (*robot.result.xmlelementhandlers.ErrorMessageHandler class method*), 526
register() (*robot.result.xmlelementhandlers.ErrorsHandler class method*), 526
register() (*robot.result.xmlelementhandlers.ForHandler class method*), 520
register() (*robot.result.xmlelementhandlers.IfHandler class method*), 521
register() (*robot.result.xmlelementhandlers.IterationHandler class method*), 521
register() (*robot.result.xmlelementhandlers.KeywordHandler class method*), 520
register() (*robot.result.xmlelementhandlers.MessageHandler class method*), 523
register() (*robot.result.xmlelementhandlers.MetadataHandler class method*), 523
register() (*robot.result.xmlelementhandlers.MetadataItemHandler class method*), 524
register() (*robot.result.xmlelementhandlers.MetaHandler class method*), 524
register() (*robot.result.xmlelementhandlers.PatternHandler class method*), 522
register() (*robot.result.xmlelementhandlers.ReturnHandler class method*), 522
register() (*robot.result.xmlelementhandlers.RobotHandler class method*), 519
register() (*robot.result.xmlelementhandlers.RootHandler class method*), 519
register() (*robot.result.xmlelementhandlers.StatisticsHandler class method*), 527
register() (*robot.result.xmlelementhandlers.StatusHandler class method*), 523
register() (*robot.result.xmlelementhandlers.SuiteHandler class method*), 520
register() (*robot.result.xmlelementhandlers.TagHandler class method*), 524
register() (*robot.result.xmlelementhandlers.TagsHandler class method*), 524
register() (*robot.result.xmlelementhandlers.TestHandler class method*), 520
register() (*robot.result.xmlelementhandlers.TimeoutHandler class method*), 525
register() (*robot.result.xmlelementhandlers.TryHandler class method*), 522
register() (*robot.result.xmlelementhandlers.ValueHandler class method*), 526
register() (*robot.result.xmlelementhandlers.VarHandler class method*), 525
register() (*robot.result.xmlelementhandlers.WhileHandler class method*), 521
register() (*robot.running.arguments.typeconverters.BooleanConverter class method*), 533
register() (*robot.running.arguments.typeconverters.ByteArrayConverter class method*), 535

register() (*robot.running.arguments.typeconverters.BytesConverter class method*), 535
register() (*robot.running.arguments.typeconverters.CombinedConverter class method*), 540
register() (*robot.running.arguments.typeconverters.CustomConverter class method*), 541
register() (*robot.running.arguments.typeconverters.DateConverter class method*), 536
register() (*robot.running.arguments.typeconverters.DateTimeConverter class method*), 536
register() (*robot.running.arguments.typeconverters.DecimalConverter class method*), 534
register() (*robot.running.arguments.typeconverters.DictionaryConverter class method*), 539
register() (*robot.running.arguments.typeconverters.EnumConverter class method*), 532
register() (*robot.running.arguments.typeconverters.FloatConverter class method*), 534
register() (*robot.running.arguments.typeconverters.FrozenSetConverter class method*), 540
register() (*robot.running.arguments.typeconverters.IntegerConverter class method*), 533
register() (*robot.running.arguments.typeconverters.ListConverter class method*), 538
register() (*robot.running.arguments.typeconverters.NoneConverter class method*), 537
register() (*robot.running.arguments.typeconverters.PathConverter class method*), 537
register() (*robot.running.arguments.typeconverters.SetConverter class method*), 539
register() (*robot.running.arguments.typeconverters.StringConverter class method*), 533
register() (*robot.running.arguments.typeconverters.TimeDeltaConverter class method*), 536
register() (*robot.running.arguments.typeconverters.TupleConverter class method*), 538
register() (*robot.running.arguments.typeconverters.TypeConverter class method*), 532
register() (*robot.running.arguments.typeconverters.TypedDictConverter class method*), 539
register() (*robot.running.model.Body class method*), 555
register() (*robot.utils.connectioncache.ConnectionCache method*), 595
register_console_logger() (*robot.output.logger.Logger method*), 308
register_error_listener() (*robot.output.logger.Logger method*), 308
register_error_listener() (*robot.output.output.Output method*), 311
register_listeners() (*robot.output.logger.Logger method*), 308
register_logger() (*robot.output.logger.Logger method*), 308
remember_run_keyword() (*in module robot.libraries.BuiltIn*), 99
remove_syslog() (*robot.output.logger.Logger method*), 308
remove_xml_logger() (*robot.output.logger.Logger method*), 308
remove_source() (*robot.reporting.jsbuildingcontext.JsBuildingContext method*), 393
release() (*robot.output.pyloggingconf.RobotHandler method*), 313
release() (*robot.running.outputcapture.PythonCapturer method*), 578
reload_library() (*robot.libraries.BuiltIn.BuiltIn method*), 84
reload_library() (*robot.running.namespace.Namespace method*), 578
Remote (*class in robot.libraries.Remote*), 128
RemoteSet (*class in robot.libraries.Remote*), 618
RemoteResult (*class in robot.libraries.Remote*), 129
RemoveMessage (*class in robot.result.keywordremover*), 449
remove() (*robot.model.body.BaseBody method*), 230
remove() (*robot.model.body.Body method*), 232
remove() (*robot.model.body.Branches method*), 234
remove() (*robot.model.itemlist.ItemList method*), 259
remove() (*robot.model.keyword.Keywords method*), 262
remove() (*robot.model.message.Messages method*), 264
remove() (*robot.model.tags.Tags method*), 277
remove() (*robot.model.testcase.TestCases method*), 285
remove() (*robot.model.testsuite.TestSuites method*), 288
remove() (*robot.result.model.Body method*), 459
remove() (*robot.result.model.Branches method*), 461
remove() (*robot.result.model.Iterations method*), 462
remove() (*robot.running.model.Body method*), 555
remove() (*robot.running.model.Imports method*), 577
remove_data_not_needed_in_report() (*robot.reporting.jsexecutionresult.JsExecutionResult method*), 394
remove_directory() (*robot.libraries.OperatingSystem.OperatingSystem method*), 117
remove_duplicates() (*robot.libraries.Collections.Collections method*), 105
remove_element() (*robot.libraries.XML.XML method*), 156
remove_element_attribute() (*robot.libraries.XML.XML method*), 155
remove_element_attributes()

(*robot.libraries.XML.XML method*), 155
remove_elements() (*robot.libraries.XML.XML method*), 156
remove_elements_attribute() (*robot.libraries.XML.XML method*), 155
remove_elements_attributes() (*robot.libraries.XML.XML method*), 155
remove_empty_suites() (*robot.model.testsuite.TestSuite method*), 287
remove_empty_suites() (*robot.result.model.TestSuite method*), 497
remove_empty_suites() (*robot.running.model.TestSuite method*), 575
remove_environment_variable() (*robot.libraries.OperatingSystem.OperatingSystem method*), 119
remove_file() (*robot.libraries.OperatingSystem.OperatingSystem method*), 117
remove_files() (*robot.libraries.OperatingSystem.OperatingSystem method*), 117
remove_from_dictionary() (*robot.libraries.Collections.Collections method*), 106
remove_from_list() (*robot.libraries.Collections.Collections method*), 106
remove_keywords (*robot.conf.settings.RebotSettings attribute*), 66
remove_keywords (*robot.conf.settings.RobotSettings attribute*), 66
remove_keywords() (*robot.result.model.TestSuite method*), 499
remove_path() (*in module robot.pythonpathsetter*), 620
remove_string() (*robot.libraries.String.String method*), 135
remove_string_using_regexp() (*robot.libraries.String.String method*), 135
remove_tags (*robot.model.configurer.SuiteConfigurer attribute*), 234
remove_tags (*robot.result.configurer.SuiteConfigurer attribute*), 409
remove_tags() (*robot.libraries.BuiltIn.BuiltIn method*), 85
remove_values_from_list() (*robot.libraries.Collections.Collections method*), 106
removeFilter() (*robot.output.pyloggingconf.RobotHandler method*), 313
RemoveKeywords (*class in robot.result.resultbuilder*), 501
repeat_keyword() (*robot.libraries.BuiltIn.BuiltIn method*), 85
replace() (*robot.running.arguments.argumentresolver.VariableReplacer method*), 529
replace_defaults() (*robot.running.arguments.argumentmapper.KeywordCallTemplate method*), 529
replace_list() (*robot.variables.replacer.VariableReplacer method*), 609
replace_list() (*robot.variables.scopes.GlobalVariables method*), 610
replace_list() (*robot.variables.scopes.VariableScopes method*), 609
replace_list() (*robot.variables.variables.Variables method*), 612
replace_scalar() (*robot.variables.replacer.VariableReplacer method*), 609
replace_scalar() (*robot.variables.scopes.GlobalVariables method*), 610
replace_scalar() (*robot.variables.scopes.VariableScopes method*), 610
replace_scalar() (*robot.variables.variables.Variables method*), 612
replace_string() (*robot.libraries.String.String method*), 135
replace_string() (*robot.variables.replacer.VariableReplacer method*), 609
replace_string() (*robot.variables.scopes.GlobalVariables method*), 610
replace_string() (*robot.variables.scopes.VariableScopes method*), 610
replace_string() (*robot.variables.variables.Variables method*), 612
replace_string_using_regexp() (*robot.libraries.String.String method*), 135
replace_variables() (*robot.libraries.BuiltIn.BuiltIn method*), 85
replace_variables() (*robot.running.timeouts.KeywordTimeout method*), 548
replace_variables() (*robot.running.timeouts.TestTimeout method*), 548
report (*robot.conf.settings.RebotSettings attribute*), 66
report (*robot.conf.settings.RobotSettings attribute*), 66
report() (*robot.output.console.dotted.StatusReporter method*), 299
report_config (*robot.conf.settings.RebotSettings attribute*), 66
report_error() (*robot.variables.resolvable.GlobalVariableValue method*), 609
report_error() (*robot.variables.resolvable.Resolvable method*), 609
report_error() (*robot.variables.tablesetter.DictVariableTableValue*

method), 612
report_error() (*robot.variables.tablesetter.ListVariableTableValue method*), 612
report_error() (*robot.variables.tablesetter.ScalarVariableTableValue method*), 612
report_error() (*robot.variables.tablesetter.VariableTableValueBase method*), 612
report_invalid_syntax() (*robot.running.model.Import method*), 576
report_invalid_syntax() (*robot.running.model.Variable method*), 576
ReportWriter (*class in robot.reporting.logreportwriters*), 395
repr_args (*robot.model.body.BodyItem attribute*), 229
repr_args (*robot.model.control.Break attribute*), 250
repr_args (*robot.model.control.Continue attribute*), 249
repr_args (*robot.model.control.For attribute*), 238
repr_args (*robot.model.control.If attribute*), 244
repr_args (*robot.model.control.IfBranch attribute*), 241
repr_args (*robot.model.control.Return attribute*), 246
repr_args (*robot.model.control.Try attribute*), 246
repr_args (*robot.model.control.TryBranch attribute*), 244
repr_args (*robot.model.control.While attribute*), 240
repr_args (*robot.model.keyword.Keyword attribute*), 259
repr_args (*robot.model.message.Message attribute*), 262
repr_args (*robot.model.modelobject.ModelObject attribute*), 265
repr_args (*robot.model.testcase.TestCase attribute*), 283
repr_args (*robot.model.testsuite.TestSuite attribute*), 285
repr_args (*robot.output.loggerhelper.Message attribute*), 311
repr_args (*robot.result.model.Break attribute*), 489
repr_args (*robot.result.model.Continue attribute*), 487
repr_args (*robot.result.model.For attribute*), 469
repr_args (*robot.result.model.ForIteration attribute*), 465
repr_args (*robot.result.model.If attribute*), 478
repr_args (*robot.result.model.IfBranch attribute*), 476
repr_args (*robot.result.model.Keyword attribute*), 492
repr_args (*robot.result.model.Message attribute*), 464
repr_args (*robot.result.model.Return attribute*), 485
repr_args (*robot.result.model.TestCase attribute*), 495
repr_args (*robot.result.model.TestSuite attribute*), 497
repr_args (*robot.result.model.Try attribute*), 482
repr_args (*robot.result.model.TryBranch attribute*), 480
repr_args (*robot.result.model.While attribute*), 473
repr_args (*robot.result.model.WhileIteration attribute*), 471
repr_args (*robot.running.model.Break attribute*), 570
repr_args (*robot.running.model.Continue attribute*), 568
repr_args (*robot.running.model.For attribute*), 559
repr_args (*robot.running.model.If attribute*), 563
repr_args (*robot.running.model.IfBranch attribute*), 561
repr_args (*robot.running.model.Keyword attribute*), 557
repr_args (*robot.running.model.Return attribute*), 567
repr_args (*robot.running.model.TestCase attribute*), 571
repr_args (*robot.running.model.TestSuite attribute*), 575
repr_args (*robot.running.model.Try attribute*), 566
repr_args (*robot.running.model.TryBranch attribute*), 564
repr_args (*robot.running.model.While attribute*), 560
request() (*robot.libraries.Remote.TimeoutHTTPSTransport method*), 130
request() (*robot.libraries.Remote.TimeoutHTTPTransport method*), 129
required (*robot.running.arguments.argumentspec.ArgInfo attribute*), 530
Reserved (*class in robot.libraries.Reserved*), 130
reset() (*robot.conf.languages.Languages method*), 32
reset() (*robot.output.console.highlighting.AnsiHighlighter method*), 303
reset() (*robot.output.console.highlighting.DosHighlighter method*), 304
reset() (*robot.output.console.highlighting.NoHighlighting method*), 303
reset() (*robot.running.importer.Importer method*), 552
reset_count() (*robot.output.console.verbose.KeywordMarker method*), 304
resizable() (*robot.libraries.dialogs_py.InputDialog method*), 178
resizable() (*robot.libraries.dialogs_py.MessageDialog method*), 164
resizable() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 206
resizable() (*robot.libraries.dialogs_py.PassFailDialog method*), 220
resizable() (*robot.libraries.dialogs_py.SelectionDialog method*), 192
Resolvable (*class in robot.variables.resolvable*), 609
resolve() (*robot.running.arguments.argumentmapper.DefaultValue method*), 529

resolve() (*robot.running.arguments.argumentresolver.ArgumentResolver method*), 529
resolve() (*robot.running.arguments.argumentresolver.NamedArgumentResolver method*), 529
resolve() (*robot.running.arguments.argumentresolver.NullNamedArgumentResolver method*), 529
resolve() (*robot.running.arguments.argumentspec.ArgumentSpec method*), 530
resolve() (*robot.variables.assigner.NoReturnValueResolver method*), 606
resolve() (*robot.variables.assigner.OneReturnValueResolver method*), 607
resolve() (*robot.variables.assigner.ScalarsAndListReturnValueResolver method*), 607
resolve() (*robot.variables.assigner.ScalarsOnlyReturnValueResolver method*), 607
resolve() (*robot.variables.resolvable.GlobalVariableValue method*), 609
resolve() (*robot.variables.resolvable.Resolvable method*), 609
resolve() (*robot.variables.tablesetter.DictVariableTableValue method*), 612
resolve() (*robot.variables.tablesetter.ListVariableTableValue method*), 612
resolve() (*robot.variables.tablesetter.ScalarVariableTableValue method*), 612
resolve() (*robot.variables.tablesetter.VariableTableValueBase method*), 612
resolve_alias_or_index() (*robot.utils.connectioncache.ConnectionCache method*), 596
resolve_arguments() (*robot.running.handlers.EmbeddedArgumentsHandler method*), 551
resolve_base() (*robot.variables.search.VariableMatch method*), 611
resolve_delayed() (*robot.variables.scopes.GlobalVariables method*), 610
resolve_delayed() (*robot.variables.scopes.VariableScopes method*), 610
resolve_delayed() (*robot.variables.store.VariableStore method*), 611
resolve_delayed() (*robot.variables.variables.Variables method*), 612
resolve_delayed_message() (*robot.output.loggerhelper.Message method*), 309
RESOURCE (*robot.parsing.lexer.tokens.END attribute*), 339
RESOURCE (*robot.parsing.lexer.tokens.EOS attribute*), 337
RESOURCE (*robot.parsing.lexer.tokens.Token attribute*), 334
resource (*robot.running.model.TestSuite attribute*), 572
resource() (*robot.running.model.Imports method*), 577
resource_setting (*robot.conf.languages.Bg attribute*), 59
resource_setting (*robot.conf.languages.Bs attribute*), 38
resource_setting (*robot.conf.languages.Cs attribute*), 35
resource_setting (*robot.conf.languages.De attribute*), 42
resource_setting (*robot.conf.languages.En attribute*), 34
resource_setting (*robot.conf.languages.Es attribute*), 51
resource_setting (*robot.conf.languages.Fi attribute*), 40
resource_setting (*robot.conf.languages.Fr attribute*), 41
resource_setting (*robot.conf.languages.Hi attribute*), 63
resource_setting (*robot.conf.languages.It attribute*), 62
resource_setting (*robot.conf.languages.Language attribute*), 32
resource_setting (*robot.conf.languages.Nl attribute*), 37
resource_setting (*robot.conf.languages.Pl attribute*), 48
resource_setting (*robot.conf.languages.Pt attribute*), 45
resource_setting (*robot.conf.languages.PtBr attribute*), 44
resource_setting (*robot.conf.languages.Ro attribute*), 61
resource_setting (*robot.conf.languages.Ru attribute*), 52
resource_setting (*robot.conf.languages.Sv attribute*), 58
resource_setting (*robot.conf.languages.Th attribute*), 47
resource_setting (*robot.conf.languages.Tr attribute*), 56
resource_setting (*robot.conf.languages.Uk attribute*), 49
resource_setting (*robot.conf.languages.ZhCn attribute*), 54
resource_setting (*robot.conf.languages.ZhTw attribute*), 55
ResourceBuilder (*class in*

robot.running.builder.transformers), 544
ResourceDocBuilder (*class in robot.libdocpkg.robotbuilder*), 72
ResourceFile (*class in robot.running.model*), 576
ResourceFileBuilder (*class in robot.running.builder.builders*), 542
ResourceFileContext (*class in robot.parsing.lexer.context*), 324
ResourceFileSettings (*class in robot.parsing.lexer.settings*), 326
ResourceImport (*class in robot.parsing.model.statements*), 351
RestParser (*class in robot.running.builder.parsers*), 543
Result (*class in robot.result.executionresult*), 413
result (*robot.reporting.resultwriter.Results attribute*), 401
result (*robot.running.modelcombiner.ModelCombiner attribute*), 577
result_config (*robot.libraries.Process.ProcessConfiguration attribute*), 128
Results (*class in robot.reporting.resultwriter*), 401
ResultVisitor (*class in robot.result.visitor*), 513
ResultWriter (*class in robot.reporting.resultwriter*), 400
Return (*class in robot.model.control*), 246
Return (*class in robot.parsing.model.statements*), 369
Return (*class in robot.result.model*), 483
Return (*class in robot.running.model*), 566
RETURN (*robot.model.body.BodyItem attribute*), 228
RETURN (*robot.model.control.Break attribute*), 249
RETURN (*robot.model.control.Continue attribute*), 248
RETURN (*robot.model.control.For attribute*), 239
RETURN (*robot.model.control.If attribute*), 243
RETURN (*robot.model.control.IfBranch attribute*), 242
RETURN (*robot.model.control.Return attribute*), 247
RETURN (*robot.model.control.Try attribute*), 246
RETURN (*robot.model.control.TryBranch attribute*), 244
RETURN (*robot.model.control.While attribute*), 240
RETURN (*robot.model.keyword.Keyword attribute*), 260
RETURN (*robot.model.message.Message attribute*), 263
RETURN (*robot.output.loggerhelper.Message attribute*), 310
RETURN (*robot.parsing.lexer.tokens.END attribute*), 339
RETURN (*robot.parsing.lexer.tokens.EOS attribute*), 337
RETURN (*robot.parsing.lexer.tokens.Token attribute*), 334
RETURN (*robot.result.model.Break attribute*), 488
RETURN (*robot.result.model.Continue attribute*), 486
RETURN (*robot.result.model.For attribute*), 467
RETURN (*robot.result.model.ForIteration attribute*), 465
RETURN (*robot.result.model.If attribute*), 477
RETURN (*robot.result.model.IfBranch attribute*), 474
RETURN (*robot.result.model.Keyword attribute*), 491
RETURN (*robot.result.model.Message attribute*), 463
RETURN (*robot.result.model.Return attribute*), 484
RETURN (*robot.result.model.Try attribute*), 481
RETURN (*robot.result.model.TryBranch attribute*), 479
RETURN (*robot.result.model.While attribute*), 472
RETURN (*robot.result.model.WhileIteration attribute*), 470
RETURN (*robot.running.model.Break attribute*), 569
RETURN (*robot.running.model.Continue attribute*), 568
RETURN (*robot.running.model.For attribute*), 558
RETURN (*robot.running.model.If attribute*), 562
RETURN (*robot.running.model.IfBranch attribute*), 561
RETURN (*robot.running.model.Keyword attribute*), 556
RETURN (*robot.running.model.Return attribute*), 566
RETURN (*robot.running.model.Try attribute*), 565
RETURN (*robot.running.model.TryBranch attribute*), 563
RETURN (*robot.running.model.While attribute*), 559
return_class (*robot.model.body.BaseBody attribute*), 229
return_class (*robot.model.body.Body attribute*), 232
return_class (*robot.model.body.Branches attribute*), 234
return_class (*robot.result.model.Body attribute*), 459
return_class (*robot.result.model.Branches attribute*), 461
return_class (*robot.result.model.Iterations attribute*), 462
return_class (*robot.running.model.Body attribute*), 555
return_code (*robot.result.executionresult.CombinedResult attribute*), 414
return_code (*robot.result.executionresult.Result attribute*), 413
return_from_keyword() (*robot.libraries.BuiltIn.BuiltIn method*), 85
return_from_keyword_if() (*robot.libraries.BuiltIn.BuiltIn method*), 86
RETURN_SETTING (*robot.parsing.lexer.tokens.END attribute*), 339
RETURN_SETTING (*robot.parsing.lexer.tokens.EOS attribute*), 337
RETURN_SETTING (*robot.parsing.lexer.tokens.Token attribute*), 334
RETURN_STATEMENT (*robot.parsing.lexer.tokens.END attribute*), 339
RETURN_STATEMENT (*robot.parsing.lexer.tokens.EOS attribute*), 337
RETURN_STATEMENT (*robot.parsing.lexer.tokens.Token attribute*), 335
ReturnFromKeyword, 618
ReturnHandler (*class in robot.result.xmlelementhandlers*), 522

ReturnLexer (*class in robot.parsing.lexer.statementlexers*), 332
ReturnStatement (*class in robot.parsing.model.statements*), 382
ReturnValueResolver() (*in module robot.variables.assigner*), 606
reverse() (*robot.model.body.BaseBody method*), 230
reverse() (*robot.model.body.Body method*), 232
reverse() (*robot.model.body.Branches method*), 234
reverse() (*robot.model.itemlist.ItemList method*), 259
reverse() (*robot.model.keyword.Keywords method*), 262
reverse() (*robot.model.message.Messages method*), 264
reverse() (*robot.model.testcase.TestCases method*), 285
reverse() (*robot.model.testsuite.TestSuites method*), 288
reverse() (*robot.result.model.Body method*), 459
reverse() (*robot.result.model.Branches method*), 461
reverse() (*robot.result.model.Iterations method*), 462
reverse() (*robot.running.model.Body method*), 555
reverse() (*robot.running.model.Imports method*), 577
reverse_list() (*robot.libraries.Collections.Collections method*), 106
Ro (*class in robot.conf.languages*), 60
robot (*module*), 7
robot() (*robot.model.tags.Tags method*), 277
robot.api (*module*), 5, 10
robot.api.deco (*module*), 11
robot.api.exceptions (*module*), 12
robot.api.logger (*module*), 14
robot.api.parsing (*module*), 15
robot.conf (*module*), 23
robot.conf.gatherfailed (*module*), 23
robot.conf.languages (*module*), 32
robot.conf.settings (*module*), 65
robot.errors (*module*), 613
robot.htmldata (*module*), 67
robot.htmldata.htmlfilewriter (*module*), 67
robot.htmldata.jsonwriter (*module*), 68
robot.htmldata.template (*module*), 68
robot.libdoc (*module*), 618
robot.libdocpkg (*module*), 69
robot.libdocpkg.builder (*module*), 69
robot.libdocpkg.consoleviewer (*module*), 69
robot.libdocpkg.datatypes (*module*), 70
robot.libdocpkg.htmlutils (*module*), 70
robot.libdocpkg.htmlwriter (*module*), 71
robot.libdocpkg.jsonbuilder (*module*), 71
robot.libdocpkg.jsonwriter (*module*), 71
robot.libdocpkg.model (*module*), 71
robot.libdocpkg.output (*module*), 72
robot.libdocpkg.robotbuilder (*module*), 72
robot.libdocpkg.standardtypes (*module*), 72
robot.libdocpkg.writer (*module*), 72
robot.libdocpkg.xmlbuilder (*module*), 73
robot.libdocpkg.xmlwriter (*module*), 73
robot.libraries (*module*), 73
robot.libraries.BuiltIn (*module*), 73
robot.libraries.Collections (*module*), 100
robot.libraries.DateTime (*module*), 106
robot.libraries.Dialogs (*module*), 111
robot.libraries.dialogs_py (*module*), 157
robot.libraries.Easter (*module*), 112
robot.libraries.OperatingSystem (*module*), 112
robot.libraries.Process (*module*), 122
robot.libraries.Remote (*module*), 128
robot.libraries.Reserved (*module*), 130
robot.libraries.Screenshot (*module*), 130
robot.libraries.String (*module*), 131
robot.libraries.Telnet (*module*), 137
robot.libraries.XML (*module*), 147
robot.model (*module*), 227
robot.model.body (*module*), 228
robot.model.configurer (*module*), 234
robot.model.control (*module*), 238
robot.model.filter (*module*), 250
robot.model.fixture (*module*), 258
robot.model.itemlist (*module*), 259
robot.model.keyword (*module*), 259
robot.model.message (*module*), 262
robot.model.metadata (*module*), 264
robot.model.modelobject (*module*), 265
robot.model.modifier (*module*), 265
robot.model.namepatterns (*module*), 270
robot.model.statistics (*module*), 270
robot.model.stats (*module*), 274
robot.model.suitestatistics (*module*), 276
robot.model.tags (*module*), 277
robot.model.tagsetter (*module*), 277
robot.model.tagstatistics (*module*), 282
robot.model.testcase (*module*), 282
robot.model.testsuite (*module*), 285
robot.model.totalstatistics (*module*), 288
robot.model.visitor (*module*), 293
robot.output (*module*), 298
robot.output.console (*module*), 298
robot.output.console.dotted (*module*), 298
robot.output.console.highlighting (*module*), 303
robot.output.console.quiet (*module*), 304
robot.output.console.verbose (*module*), 304
robot.output.debugfile (*module*), 305
robot.output.filelogger (*module*), 305
robot.output.librarylogger (*module*), 305

robot.output.listenerarguments (*module*), 306
robot.output.listenermethods (*module*), 307
robot.output.listeners (*module*), 307
robot.output.logger (*module*), 308
robot.output.loggerhelper (*module*), 309
robot.output.output (*module*), 311
robot.output.pyloggingconf (*module*), 312
robot.output.stdoutlogsplitter (*module*), 313
robot.output.xmllogger (*module*), 313
robot.parsing (*module*), 318
robot.parsing.lexer (*module*), 318
robot.parsing.lexer.blocklexers (*module*), 318
robot.parsing.lexer.context (*module*), 323
robot.parsing.lexer.lexer (*module*), 325
robot.parsing.lexer.settings (*module*), 326
robot.parsing.lexer.statementlexers (*module*), 327
robot.parsing.lexer.tokenizer (*module*), 333
robot.parsing.lexer.tokens (*module*), 333
robot.parsing.model (*module*), 340
robot.parsing.model.blocks (*module*), 340
robot.parsing.model.statements (*module*), 345
robot.parsing.model.visitor (*module*), 388
robot.parsing.parser (*module*), 389
robot.parsing.parser.blockparsers (*module*), 389
robot.parsing.parser.fileparser (*module*), 390
robot.parsing.parser.parser (*module*), 391
robot.parsing.suitestructure (*module*), 392
robot.pythonpathsetter (*module*), 620
robot.rebot (*module*), 620
robot.reporting (*module*), 393
robot.reporting.expandkeywordmatcher (*module*), 393
robot.reporting.jsbuildingcontext (*module*), 393
robot.reporting.jsexecutionresult (*module*), 394
robot.reporting.jsmodelbuilders (*module*), 394
robot.reporting.jswriter (*module*), 395
robot.reporting.logreportwriters (*module*), 395
robot.reporting.outputwriter (*module*), 395
robot.reporting.resultwriter (*module*), 400
robot.reporting.stringcache (*module*), 401
robot.reporting.xunitwriter (*module*), 402
robot.result (*module*), 407
robot.result.configurer (*module*), 408
robot.result.executionerrors (*module*), 412
robot.result.executionresult (*module*), 413
robot.result.flattenkeywordmatcher (*module*), 415
robot.result.keywordremover (*module*), 415
robot.result.merger (*module*), 449
robot.result.messagefilter (*module*), 453
robot.result.model (*module*), 457
robot.result.modeldeprecation (*module*), 499
robot.result.resultbuilder (*module*), 500
robot.result.suiteteardownfailed (*module*), 505
robot.result.visitor (*module*), 513
robot.result.xmlelementhandlers (*module*), 519
robot.run (*module*), 621
robot.running (*module*), 527
robot.running.arguments (*module*), 528
robot.running.arguments.argumentconverter (*module*), 528
robot.running.arguments.argumentmapper (*module*), 528
robot.running.arguments.argumentparser (*module*), 529
robot.running.arguments.argumentresolver (*module*), 529
robot.running.arguments.argumentspec (*module*), 530
robot.running.arguments.argumentvalidator (*module*), 531
robot.running.arguments.customconverters (*module*), 531
robot.running.arguments.embedded (*module*), 531
robot.running.arguments.typeconverters (*module*), 531
robot.running.arguments.typevalidator (*module*), 541
robot.running.bodyrunner (*module*), 549
robot.running.builder (*module*), 541
robot.running.builder.builders (*module*), 541
robot.running.builder.parsers (*module*), 542
robot.running.builder.settings (*module*), 543
robot.running.builder.transformers (*module*), 544
robot.running.context (*module*), 550
robot.running.dynamicmethods (*module*), 550
robot.running.handlers (*module*), 551
robot.running.handlerstore (*module*), 551

robot.running.importer (*module*), 552
robot.running.librarykeywordrunner (*module*), 552
robot.running.libraryscopes (*module*), 553
robot.running.model (*module*), 553
robot.running.modelcombiner (*module*), 577
robot.running.namespace (*module*), 577
robot.running.outputcapture (*module*), 578
robot.running.randomizer (*module*), 578
robot.running.runkwregister (*module*), 583
robot.running.signalhandler (*module*), 583
robot.running.status (*module*), 583
robot.running.statusreporter (*module*), 584
robot.running.suiterunner (*module*), 585
robot.running.testlibraries (*module*), 589
robot.running.timeouts (*module*), 548
robot.running.timeouts.posix (*module*), 549
robot.running.timeouts.windows (*module*), 549
robot.running.usererrorhandler (*module*), 589
robot.running.userkeyword (*module*), 589
robot.running.userkeywordrunner (*module*), 590
robot.testdoc (*module*), 623
robot.utils (*module*), 591
robot.utils.application (*module*), 591
robot.utils.argumentparser (*module*), 592
robot.utils.asserts (*module*), 593
robot.utils.charwidth (*module*), 595
robot.utils.compress (*module*), 595
robot.utils.connectioncache (*module*), 595
robot.utils.dotdict (*module*), 596
robot.utils.encoding (*module*), 597
robot.utils.encodingsniffer (*module*), 597
robot.utils.error (*module*), 598
robot.utils.escaping (*module*), 598
robot.utils.etreewrapper (*module*), 598
robot.utils.filereader (*module*), 598
robot.utils.frange (*module*), 599
robot.utils.htmlformatters (*module*), 599
robot.utils.importer (*module*), 600
robot.utils.markuputils (*module*), 602
robot.utils.markupwriters (*module*), 602
robot.utils.match (*module*), 603
robot.utils.misc (*module*), 603
robot.utils.normalizing (*module*), 604
robot.utils.platform (*module*), 605
robot.utils.recommendations (*module*), 605
robot.variables (*module*), 606
robot.variables.assigner (*module*), 606
robot.variables.evaluation (*module*), 607
robot.variables.filesetter (*module*), 607
robot.variables.finders (*module*), 608
robot.variables.notfound (*module*), 608
robot.variables.replacer (*module*), 609
robot.variables.resolvable (*module*), 609
robot.variables.scopes (*module*), 609
robot.variables.search (*module*), 611
robot.variables.store (*module*), 611
robot.variables.tablesetter (*module*), 612
robot.variables.variables (*module*), 612
robot.version (*module*), 625
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*), 312
ROBOT_LIBRARY_SCOPE (*robot.libraries.BuiltIn.BuiltIn attribute*), 76
ROBOT_LIBRARY_SCOPE (*robot.libraries.Collections.Collections attribute*), 101
ROBOT_LIBRARY_SCOPE (*robot.libraries.OperatingSystem.OperatingSystem attribute*), 113
ROBOT_LIBRARY_SCOPE (*robot.libraries.Process.Process attribute*), 125
ROBOT_LIBRARY_SCOPE (*robot.libraries.Remote.Remote attribute*), 128
ROBOT_LIBRARY_SCOPE (*robot.libraries.Reserved.Reserved attribute*), 130
ROBOT_LIBRARY_SCOPE (*robot.libraries.Screenshot.Screenshot attribute*), 131
ROBOT_LIBRARY_SCOPE (*robot.libraries.String.String attribute*), 132
ROBOT_LIBRARY_SCOPE (*robot.libraries.Telnet.Telnet attribute*), 140
ROBOT_LIBRARY_SCOPE (*robot.libraries.XML.XML attribute*), 150
ROBOT_LIBRARY_VERSION (*robot.libraries.BuiltIn.BuiltIn attribute*), 76
ROBOT_LIBRARY_VERSION (*robot.libraries.Collections.Collections attribute*), 101
ROBOT_LIBRARY_VERSION (*robot.libraries.OperatingSystem.OperatingSystem attribute*), 113
ROBOT_LIBRARY_VERSION

(*robot.libraries.Process.Process attribute*), 125
ROBOT_LIBRARY_VERSION (*robot.libraries.Screenshot.Screenshot attribute*), 131
ROBOT_LIBRARY_VERSION (*robot.libraries.String.String attribute*), 132
ROBOT_LIBRARY_VERSION (*robot.libraries.Telnet.Telnet attribute*), 140
ROBOT_LIBRARY_VERSION (*robot.libraries.XML.XML attribute*), 150
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*), 146
RobotError, 613
RobotFramework (*class in robot.run*), 622
RobotHandler (*class in robot.output.pyloggingconf*), 312
RobotHandler (*class in robot.result.xmlelementhandlers*), 519
RobotModelWriter (*class in robot.reporting.logreportwriters*), 395
RobotNotRunningError, 99
RobotParser (*class in robot.running.builder.parsers*), 542
RobotSettings (*class in robot.conf.settings*), 65
RootHandler (*class in robot.result.xmlelementhandlers*), 519
rowconfigure() (*robot.libraries.dialogs_py.InputDialog method*), 178
rowconfigure() (*robot.libraries.dialogs_py.MessageDialog method*), 164
rowconfigure() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 206
rowconfigure() (*robot.libraries.dialogs_py.PassFailDialog method*), 220
rowconfigure() (*robot.libraries.dialogs_py.SelectionDialog method*), 192
rpa (*robot.conf.settings.RebotSettings attribute*), 66
rpa (*robot.conf.settings.RobotSettings attribute*), 66
rpa (*robot.model.testsuite.TestSuite attribute*), 285
rpa (*robot.result.model.TestSuite attribute*), 498
rpa (*robot.running.model.TestSuite attribute*), 575
Ru (*class in robot.conf.languages*), 52
RulerFormatter (*class in robot.utils.htmlformatters*), 599
run() (*in module robot*), 7
run() (*in module robot.run*), 622
run() (*robot.libraries.OperatingSystem.OperatingSystem method*), 113
run() (*robot.running.bodyrunner.BodyRunner method*), 549
run() (*robot.running.bodyrunner.ForInEnumerateRunner method*), 549
run() (*robot.running.bodyrunner.ForInRangeRunner method*), 549
run() (*robot.running.bodyrunner.ForInRunner method*), 549
run() (*robot.running.bodyrunner.ForInZipRunner method*), 549
run() (*robot.running.bodyrunner.IfRunner method*), 550
run() (*robot.running.bodyrunner.KeywordRunner method*), 549
run() (*robot.running.bodyrunner.TryRunner method*), 550
run() (*robot.running.bodyrunner.WhileRunner method*), 549
run() (*robot.running.librarykeywordrunner.EmbeddedArgumentsRunner method*), 552
run() (*robot.running.librarykeywordrunner.LibraryKeywordRunner method*), 552
run() (*robot.running.librarykeywordrunner.RunKeywordRunner method*), 553
run() (*robot.running.model.Break method*), 569
run() (*robot.running.model.Continue method*), 567
run() (*robot.running.model.For method*), 557
run() (*robot.running.model.If method*), 562
run() (*robot.running.model.Keyword method*), 555
run() (*robot.running.model.Return method*), 566
run() (*robot.running.model.TestSuite method*), 573
run() (*robot.running.model.Try method*), 564
run() (*robot.running.model.While method*), 559
run() (*robot.running.timeouts.KeywordTimeout method*), 548
run() (*robot.running.timeouts.TestTimeout method*), 548
run() (*robot.running.usererrorhandler.UserErrorHandler method*), 589
run() (*robot.running.userkeywordrunner.EmbeddedArgumentsRunner method*), 590
run() (*robot.running.userkeywordrunner.UserKeywordRunner*

method), 590
run_and_return_rc() (*robot.libraries.OperatingSystem.OperatingSystem method*), 114
run_and_return_rc_and_output() (*robot.libraries.OperatingSystem.OperatingSystem method*), 114
run_cli() (*in module robot*), 8
run_cli() (*in module robot.run*), 622
run_empty_suite (*robot.conf.settings.RobotSettings attribute*), 65
run_keyword() (*robot.libraries.BuiltIn.BuiltIn method*), 86
run_keyword() (*robot.libraries.Remote.Remote method*), 128
run_keyword() (*robot.libraries.Remote.XmlRpcRemoteClient method*), 129
run_keyword_and_continue_on_failure() (*robot.libraries.BuiltIn.BuiltIn method*), 86
run_keyword_and_expect_error() (*robot.libraries.BuiltIn.BuiltIn method*), 86
run_keyword_and_ignore_error() (*robot.libraries.BuiltIn.BuiltIn method*), 87
run_keyword_and_return() (*robot.libraries.BuiltIn.BuiltIn method*), 87
run_keyword_and_return_if() (*robot.libraries.BuiltIn.BuiltIn method*), 87
run_keyword_and_return_status() (*robot.libraries.BuiltIn.BuiltIn method*), 87
run_keyword_and_warn_on_failure() (*robot.libraries.BuiltIn.BuiltIn method*), 87
run_keyword_if() (*robot.libraries.BuiltIn.BuiltIn method*), 87
run_keyword_if_all_tests_passed() (*robot.libraries.BuiltIn.BuiltIn method*), 88
run_keyword_if_any_tests_failed() (*robot.libraries.BuiltIn.BuiltIn method*), 88
run_keyword_if_test_failed() (*robot.libraries.BuiltIn.BuiltIn method*), 88
run_keyword_if_test_passed() (*robot.libraries.BuiltIn.BuiltIn method*), 88
run_keyword_if_timeout_occurred() (*robot.libraries.BuiltIn.BuiltIn method*), 88
run_keyword_unless() (*robot.libraries.BuiltIn.BuiltIn method*), 89
run_keyword_variant() (*in module robot.libraries.BuiltIn*), 73
run_keywords() (*robot.libraries.BuiltIn.BuiltIn method*), 89
run_process() (*robot.libraries.Process.Process method*), 125
RunKeyword (*class in robot.running.dynamicmethods*), 550
RunKeywordRunner (*class in robot.running.librarykeywordrunner*), 552

S

save() (*robot.libdocpkg.model.LibraryDoc method*), 71
save() (*robot.parsing.model.blocks.File method*), 341
save() (*robot.result.executionresult.CombinedResult method*), 414
save() (*robot.result.executionresult.Result method*), 414
save_xml() (*robot.libraries.XML.XML method*), 157
ScalarsAndListReturnValueResolver (*class in robot.variables.assigner*), 607
ScalarsOnlyReturnValueResolver (*class in robot.variables.assigner*), 607
ScalarVariableTableValue (*class in robot.variables.tablesetter*), 612
Screenshot (*class in robot.libraries.Screenshot*), 130
ScreenshotTaker (*class in robot.libraries.Screenshot*), 131
search() (*robot.libdocpkg.consoleviewer.KeywordMatcher method*), 70
search_variable() (*in module robot.variables.search*), 611
Section (*class in robot.parsing.model.blocks*), 341
SectionHeader (*class in robot.parsing.model.statements*), 349
SectionHeaderLexer (*class in robot.parsing.lexer.statementlexers*), 328
SectionLexer (*class in robot.parsing.lexer.blocklexers*), 319
SectionParser (*class in robot.parsing.parser.fileparser*), 390
selection_clear() (*robot.libraries.dialogs_py.InputDialog method*), 179
selection_clear() (*robot.libraries.dialogs_py.MessageDialog method*), 165
selection_clear() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 207

selection_clear() (*robot.libraries.dialogs_py.PassFailDialog method*), 221
selection_clear() (*robot.libraries.dialogs_py.SelectionDialog method*), 193
selection_get() (*robot.libraries.dialogs_py.InputDialog method*), 179
selection_get() (*robot.libraries.dialogs_py.MessageDialog method*), 165
selection_get() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 207
selection_get() (*robot.libraries.dialogs_py.PassFailDialog method*), 221
selection_get() (*robot.libraries.dialogs_py.SelectionDialog method*), 193
selection_handle() (*robot.libraries.dialogs_py.InputDialog method*), 179
selection_handle() (*robot.libraries.dialogs_py.MessageDialog method*), 165
selection_handle() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 207
selection_handle() (*robot.libraries.dialogs_py.PassFailDialog method*), 221
selection_handle() (*robot.libraries.dialogs_py.SelectionDialog method*), 193
selection_own() (*robot.libraries.dialogs_py.InputDialog method*), 179
selection_own() (*robot.libraries.dialogs_py.MessageDialog method*), 165
selection_own() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 207
selection_own() (*robot.libraries.dialogs_py.PassFailDialog method*), 221
selection_own() (*robot.libraries.dialogs_py.SelectionDialog method*), 193
selection_own_get() (*robot.libraries.dialogs_py.InputDialog method*), 179
selection_own_get() (*robot.libraries.dialogs_py.MessageDialog method*), 165
selection_own_get() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 207
selection_own_get() (*robot.libraries.dialogs_py.PassFailDialog method*), 221
selection_own_get() (*robot.libraries.dialogs_py.SelectionDialog method*), 193
SelectionDialog (*class in robot.libraries.dialogs_py*), 185
send() (*robot.libraries.dialogs_py.InputDialog method*), 179
send() (*robot.libraries.dialogs_py.MessageDialog method*), 165
send() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 207
send() (*robot.libraries.dialogs_py.PassFailDialog method*), 221
send() (*robot.libraries.dialogs_py.SelectionDialog method*), 193
send_content() (*robot.libraries.Remote.TimeoutHTTPSTransport method*), 130
send_content() (*robot.libraries.Remote.TimeoutHTTPTransport method*), 129
send_headers() (*robot.libraries.Remote.TimeoutHTTPSTransport method*), 130
send_headers() (*robot.libraries.Remote.TimeoutHTTPTransport method*), 129
send_request() (*robot.libraries.Remote.TimeoutHTTPSTransport method*), 130
send_request() (*robot.libraries.Remote.TimeoutHTTPTransport method*), 129
send_signal_to_process() (*robot.libraries.Process.Process method*), 126
SEPARATOR (*robot.parsing.lexer.tokens.END attribute*), 339
SEPARATOR (*robot.parsing.lexer.tokens.EOS attribute*), 337
SEPARATOR (*robot.parsing.lexer.tokens.Token attribute*), 335
separator (*robot.parsing.lexer.statementlexers.ForHeaderLexer attribute*), 331
seq2str() (*in module robot.utils.misc*), 604
seq2str2() (*in module robot.utils.misc*), 604
set() (*robot.result.keywordremover.RemovalMessage method*), 449
set() (*robot.variables.filesetter.VariableFileSetter method*), 607
set() (*robot.variables.tablesetter.VariableTableSetter method*), 612
set_debuglevel() (*robot.libraries.Telnet.TelnetConnection method*), 146
set_default_log_level() (*robot.libraries.Telnet.TelnetConnection method*), 142
set_earlier_failures() (*robot.errors.BreakLoop method*), 618
set_earlier_failures() (*robot.errors.ContinueLoop method*), 617

set_earlier_failures() (*robot.errors.ExecutionPassed method*), 616
set_earlier_failures() (*robot.errors.PassExecution method*), 617
set_earlier_failures() (*robot.errors.ReturnFromKeyword method*), 618
set_element_attribute() (*robot.libraries.XML.XML method*), 154
set_element_tag() (*robot.libraries.XML.XML method*), 154
set_element_text() (*robot.libraries.XML.XML method*), 154
set_elements_attribute() (*robot.libraries.XML.XML method*), 155
set_elements_tag() (*robot.libraries.XML.XML method*), 154
set_elements_text() (*robot.libraries.XML.XML method*), 154
set_encoding() (*robot.libraries.Telnet.TelnetConnection method*), 142
set_environment_variable() (*robot.libraries.OperatingSystem.OperatingSystem method*), 119
set_error() (*robot.parsing.lexer.tokens.END method*), 340
set_error() (*robot.parsing.lexer.tokens.EOS method*), 338
set_error() (*robot.parsing.lexer.tokens.Token method*), 335
set_execution_mode() (*robot.result.executionresult.CombinedResult method*), 414
set_execution_mode() (*robot.result.executionresult.Result method*), 414
set_from_file() (*robot.variables.scopes.GlobalVariables method*), 610
set_from_file() (*robot.variables.scopes.VariableScopes method*), 610
set_from_file() (*robot.variables.variables.Variables method*), 613
set_from_variable_table() (*robot.variables.scopes.GlobalVariables method*), 610
set_from_variable_table() (*robot.variables.scopes.VariableScopes method*), 610
set_from_variable_table() (*robot.variables.variables.Variables method*), 613
set_global() (*robot.variables.scopes.SetVariables method*), 610
set_global() (*robot.variables.scopes.VariableScopes method*), 610
set_global_variable() (*robot.libraries.BuiltIn.BuiltIn method*), 89
set_if_removed() (*robot.result.keywordremover.RemovalMessage method*), 449
set_keyword() (*robot.variables.scopes.SetVariables method*), 610
set_keyword() (*robot.variables.scopes.VariableScopes method*), 610
set_keyword_timeout() (*robot.running.timeouts.TestTimeout method*), 548
set_level() (*in module robot.output.pyloggingconf*), 312
set_level() (*robot.output.filelogger.FileLogger method*), 305
set_level() (*robot.output.logger.Logger method*), 309
set_level() (*robot.output.loggerhelper.AbstractLogger method*), 309
set_level() (*robot.output.loggerhelper.IsLogged method*), 311
set_level() (*robot.output.output.Output method*), 311
set_library_search_order() (*robot.libraries.BuiltIn.BuiltIn method*), 89
set_list_value() (*robot.libraries.Collections.Collections method*), 106
set_local_variable() (*robot.libraries.BuiltIn.BuiltIn method*), 90
set_local_variable() (*robot.variables.scopes.VariableScopes method*), 610
set_log_level() (*robot.libraries.BuiltIn.BuiltIn method*), 90
set_log_level() (*robot.output.listeners.LibraryListeners method*), 307
set_log_level() (*robot.output.listeners.Listeners method*), 307
set_log_level() (*robot.output.output.Output method*), 311
set_log_level() (*robot.output.xmllogger.XmlLogger method*), 313
set_log_level() (*robot.reporting.outputwriter.OutputWriter method*), 397
set_modified_time() (*robot.libraries.OperatingSystem.OperatingSystem method*), 121
set_name() (*robot.output.pyloggingconf.RobotHandler method*), 313

set_newline() (*robot.libraries.Telnet.TelnetConnection method*), 142
set_option_negotiation_callback() (*robot.libraries.Telnet.TelnetConnection method*), 146
set_prompt() (*robot.libraries.Telnet.TelnetConnection method*), 142
set_screenshot_directory() (*robot.libraries.Screenshot.Screenshot method*), 131
set_search_order() (*robot.running.namespace.Namespace method*), 577
set_suite() (*robot.variables.scopes.SetVariables method*), 610
set_suite() (*robot.variables.scopes.VariableScopes method*), 610
set_suite_documentation() (*robot.libraries.BuiltIn.BuiltIn method*), 90
set_suite_metadata() (*robot.libraries.BuiltIn.BuiltIn method*), 90
set_suite_variable() (*robot.libraries.BuiltIn.BuiltIn method*), 90
set_tags() (*robot.libraries.BuiltIn.BuiltIn method*), 91
set_tags() (*robot.model.testsuite.TestSuite method*), 286
set_tags() (*robot.result.model.TestSuite method*), 498
set_tags() (*robot.running.model.TestSuite method*), 575
set_task_variable() (*robot.libraries.BuiltIn.BuiltIn method*), 91
set_telnetlib_log_level() (*robot.libraries.Telnet.TelnetConnection method*), 142
set_test() (*robot.variables.scopes.SetVariables method*), 610
set_test() (*robot.variables.scopes.VariableScopes method*), 610
set_test_documentation() (*robot.libraries.BuiltIn.BuiltIn method*), 91
set_test_message() (*robot.libraries.BuiltIn.BuiltIn method*), 91
set_test_variable() (*robot.libraries.BuiltIn.BuiltIn method*), 92
set_timeout() (*robot.libraries.Telnet.TelnetConnection method*), 141
set_to_dictionary() (*robot.libraries.Collections.Collections method*), 106
set_variable() (*robot.libraries.BuiltIn.BuiltIn method*), 92
set_variable_if() (*robot.libraries.BuiltIn.BuiltIn method*), 92
SetConverter (*class in robot.running.arguments.typeconverters*), 539
setdefault() (*robot.model.metadata.Metadata method*), 265
setdefault() (*robot.utils.dotdict.DotDict method*), 597
setdefault() (*robot.utils.normalizing.NormalizedDict method*), 605
setdefault() (*robot.variables.evaluation.EvaluationNamespace method*), 607
setFormatter() (*robot.output.pyloggingconf.RobotHandler method*), 313
SetLanguages (*class in robot.parsing.parser.parser*), 392
setLevel() (*robot.output.pyloggingconf.RobotHandler method*), 313
setter() (*robot.utils.misc.classproperty method*), 604
SETTING_HEADER (*robot.parsing.lexer.tokens.END attribute*), 339
SETTING_HEADER (*robot.parsing.lexer.tokens.EOS attribute*), 337
SETTING_HEADER (*robot.parsing.lexer.tokens.Token attribute*), 333
setting_section() (*robot.parsing.lexer.context.FileContext method*), 323
setting_section() (*robot.parsing.lexer.context.InitFileContext method*), 325
setting_section() (*robot.parsing.lexer.context.ResourceFileContext method*), 324
setting_section() (*robot.parsing.lexer.context.TestCaseFileContext method*), 324
SETTING_TOKENS (*robot.parsing.lexer.tokens.END attribute*), 339
SETTING_TOKENS (*robot.parsing.lexer.tokens.EOS attribute*), 337
SETTING_TOKENS (*robot.parsing.lexer.tokens.Token attribute*), 335
SettingLexer (*class in robot.parsing.lexer.statementlexers*), 330
Settings (*class in robot.parsing.lexer.settings*), 326
settings (*robot.conf.languages.Bg attribute*), 60

settings (*robot.conf.languages.Bs attribute*), 39
settings (*robot.conf.languages.Cs attribute*), 37
settings (*robot.conf.languages.De attribute*), 43
settings (*robot.conf.languages.En attribute*), 35
settings (*robot.conf.languages.Es attribute*), 52
settings (*robot.conf.languages.Fi attribute*), 41
settings (*robot.conf.languages.Fr attribute*), 42
settings (*robot.conf.languages.Hi attribute*), 64
settings (*robot.conf.languages.It attribute*), 63
settings (*robot.conf.languages.Language attribute*), 34
settings (*robot.conf.languages.Nl attribute*), 38
settings (*robot.conf.languages.Pl attribute*), 49
settings (*robot.conf.languages.Pt attribute*), 46
settings (*robot.conf.languages.PtBr attribute*), 45
settings (*robot.conf.languages.Ro attribute*), 62
settings (*robot.conf.languages.Ru attribute*), 53
settings (*robot.conf.languages.Sv attribute*), 59
settings (*robot.conf.languages.Th attribute*), 48
settings (*robot.conf.languages.Tr attribute*), 57
settings (*robot.conf.languages.Uk attribute*), 50
settings (*robot.conf.languages.ZhCn attribute*), 55
settings (*robot.conf.languages.ZhTw attribute*), 56
settings_class (*robot.parsing.lexer.context.FileContext attribute*), 323
settings_class (*robot.parsing.lexer.context.InitFileContext attribute*), 324
settings_class (*robot.parsing.lexer.context.KeywordContext attribute*), 325
settings_class (*robot.parsing.lexer.context.LexingContext attribute*), 323
settings_class (*robot.parsing.lexer.context.ResourceFileContext attribute*), 324
settings_class (*robot.parsing.lexer.context.TestCaseContext attribute*), 325
settings_class (*robot.parsing.lexer.context.TestCaseFileContext attribute*), 324
settings_header (*robot.conf.languages.Bg attribute*), 59
settings_header (*robot.conf.languages.Bs attribute*), 38
settings_header (*robot.conf.languages.Cs attribute*), 35
settings_header (*robot.conf.languages.De attribute*), 42
settings_header (*robot.conf.languages.En attribute*), 34
settings_header (*robot.conf.languages.Es attribute*), 51
settings_header (*robot.conf.languages.Fi attribute*), 39
settings_header (*robot.conf.languages.Fr attribute*), 41
settings_header (*robot.conf.languages.Hi attribute*), 63
settings_header (*robot.conf.languages.It attribute*), 62
settings_header (*robot.conf.languages.Language attribute*), 32
settings_header (*robot.conf.languages.Nl attribute*), 37
settings_header (*robot.conf.languages.Pl attribute*), 48
settings_header (*robot.conf.languages.Pt attribute*), 45
settings_header (*robot.conf.languages.PtBr attribute*), 44
settings_header (*robot.conf.languages.Ro attribute*), 60
settings_header (*robot.conf.languages.Ru attribute*), 52
settings_header (*robot.conf.languages.Sv attribute*), 58
settings_header (*robot.conf.languages.Th attribute*), 46
settings_header (*robot.conf.languages.Tr attribute*), 56
settings_header (*robot.conf.languages.Uk attribute*), 49
settings_header (*robot.conf.languages.ZhCn attribute*), 53
settings_header (*robot.conf.languages.ZhTw attribute*), 55
SettingsBuilder (*class in robot.running.builder.transformers*), 544
SettingSection (*class in robot.parsing.model.blocks*), 341
SettingSectionHeaderLexer (*class in robot.parsing.lexer.statementlexers*), 328
SettingSectionLexer (*class in robot.parsing.lexer.blocklexers*), 319
SettingSectionParser (*class in robot.parsing.parser.fileparser*), 390
Setup (*class in robot.parsing.model.statements*), 364
SETUP (*robot.model.body.BodyItem attribute*), 228
SETUP (*robot.model.control.Break attribute*), 249
SETUP (*robot.model.control.Continue attribute*), 248
SETUP (*robot.model.control.For attribute*), 239
SETUP (*robot.model.control.If attribute*), 243
SETUP (*robot.model.control.IfBranch attribute*), 242
SETUP (*robot.model.control.Return attribute*), 247
SETUP (*robot.model.control.Try attribute*), 246
SETUP (*robot.model.control.TryBranch attribute*), 244
SETUP (*robot.model.control.While attribute*), 240
SETUP (*robot.model.keyword.Keyword attribute*), 260
setup (*robot.model.keyword.Keywords attribute*), 261
SETUP (*robot.model.message.Message attribute*), 263

setup (*robot.model.testcase.TestCase attribute*), 283
setup (*robot.model.testsuite.TestSuite attribute*), 285
SETUP (*robot.output.loggerhelper.Message attribute*), 310
SETUP (*robot.parsing.lexer.tokens.END attribute*), 339
SETUP (*robot.parsing.lexer.tokens.EOS attribute*), 337
SETUP (*robot.parsing.lexer.tokens.Token attribute*), 334
SETUP (*robot.result.model.Break attribute*), 488
SETUP (*robot.result.model.Continue attribute*), 486
SETUP (*robot.result.model.For attribute*), 467
SETUP (*robot.result.model.ForIteration attribute*), 465
SETUP (*robot.result.model.If attribute*), 477
SETUP (*robot.result.model.IfBranch attribute*), 474
SETUP (*robot.result.model.Keyword attribute*), 491
SETUP (*robot.result.model.Message attribute*), 463
SETUP (*robot.result.model.Return attribute*), 484
setup (*robot.result.model.TestCase attribute*), 495
setup (*robot.result.model.TestSuite attribute*), 498
SETUP (*robot.result.model.Try attribute*), 481
SETUP (*robot.result.model.TryBranch attribute*), 479
SETUP (*robot.result.model.While attribute*), 472
SETUP (*robot.result.model.WhileIteration attribute*), 470
setup (*robot.running.builder.settings.Defaults attribute*), 543
setup (*robot.running.builder.settings.TestSettings attribute*), 543
SETUP (*robot.running.model.Break attribute*), 569
SETUP (*robot.running.model.Continue attribute*), 568
SETUP (*robot.running.model.For attribute*), 558
SETUP (*robot.running.model.If attribute*), 562
SETUP (*robot.running.model.IfBranch attribute*), 561
SETUP (*robot.running.model.Keyword attribute*), 556
SETUP (*robot.running.model.Return attribute*), 566
setup (*robot.running.model.TestCase attribute*), 571
setup (*robot.running.model.TestSuite attribute*), 575
SETUP (*robot.running.model.Try attribute*), 565
SETUP (*robot.running.model.TryBranch attribute*), 563
SETUP (*robot.running.model.While attribute*), 559
setup_executed() (*robot.running.status.SuiteStatus method*), 583
setup_executed() (*robot.running.status.TestStatus method*), 583
setup_message (*robot.running.status.ParentMessage attribute*), 584
setup_message (*robot.running.status.SuiteMessage attribute*), 584
setup_message (*robot.running.status.TestMessage attribute*), 584
setup_setting (*robot.conf.languages.Bg attribute*), 60
setup_setting (*robot.conf.languages.Bs attribute*), 39
setup_setting (*robot.conf.languages.Cs attribute*), 36
setup_setting (*robot.conf.languages.De attribute*), 43
setup_setting (*robot.conf.languages.En attribute*), 35
setup_setting (*robot.conf.languages.Es attribute*), 51
setup_setting (*robot.conf.languages.Fi attribute*), 40
setup_setting (*robot.conf.languages.Fr attribute*), 42
setup_setting (*robot.conf.languages.Hi attribute*), 64
setup_setting (*robot.conf.languages.It attribute*), 62
setup_setting (*robot.conf.languages.Language attribute*), 33
setup_setting (*robot.conf.languages.Nl attribute*), 37
setup_setting (*robot.conf.languages.Pl attribute*), 49
setup_setting (*robot.conf.languages.Pt attribute*), 46
setup_setting (*robot.conf.languages.PtBr attribute*), 44
setup_setting (*robot.conf.languages.Ro attribute*), 61
setup_setting (*robot.conf.languages.Ru attribute*), 53
setup_setting (*robot.conf.languages.Sv attribute*), 58
setup_setting (*robot.conf.languages.Th attribute*), 47
setup_setting (*robot.conf.languages.Tr attribute*), 57
setup_setting (*robot.conf.languages.Uk attribute*), 50
setup_setting (*robot.conf.languages.ZhCn attribute*), 54
setup_setting (*robot.conf.languages.ZhTw attribute*), 55
setup_skipped_message (*robot.running.status.ParentMessage attribute*), 584
setup_skipped_message (*robot.running.status.SuiteMessage attribute*), 584
setup_skipped_message (*robot.running.status.TestMessage attribute*), 584
setvar() (*robot.libraries.dialogs_py.InputDialog method*), 179
setvar() (*robot.libraries.dialogs_py.MessageDialog method*), 165
setvar() (*robot.libraries.dialogs_py.MultipleSelectionDialog*

method), 207
setvar() (*robot.libraries.dialogs_py.PassFailDialog method*), 221
setvar() (*robot.libraries.dialogs_py.SelectionDialog method*), 193
SetVariables (*class in robot.variables.scopes*), 610
shortdoc (*robot.libdocpkg.model.KeywordDoc attribute*), 72
shortdoc (*robot.running.usererrorhandler.UserErrorHandler attribute*), 589
shortdoc (*robot.running.userkeyword.EmbeddedArgumentsHandler attribute*), 590
shortdoc (*robot.running.userkeyword.UserKeywordHandler attribute*), 590
should_be_byte_string() (*robot.libraries.String.String method*), 137
should_be_empty() (*robot.libraries.BuiltIn.BuiltIn method*), 92
should_be_equal() (*robot.libraries.BuiltIn.BuiltIn method*), 92
should_be_equal_as_integers() (*robot.libraries.BuiltIn.BuiltIn method*), 93
should_be_equal_as_numbers() (*robot.libraries.BuiltIn.BuiltIn method*), 93
should_be_equal_as_strings() (*robot.libraries.BuiltIn.BuiltIn method*), 93
should_be_lower_case() (*robot.libraries.String.String method*), 137
should_be_string() (*robot.libraries.String.String method*), 136
should_be_title_case() (*robot.libraries.String.String method*), 137
should_be_true() (*robot.libraries.BuiltIn.BuiltIn method*), 94
should_be_unicode_string() (*robot.libraries.String.String method*), 136
should_be_upper_case() (*robot.libraries.String.String method*), 137
should_contain() (*robot.libraries.BuiltIn.BuiltIn method*), 94
should_contain_any() (*robot.libraries.BuiltIn.BuiltIn method*), 94
should_contain_match() (*robot.libraries.Collections.Collections method*), 101
should_contain_x_times() (*robot.libraries.BuiltIn.BuiltIn method*), 94
should_end_with() (*robot.libraries.BuiltIn.BuiltIn method*), 95
should_exist() (*robot.libraries.OperatingSystem.OperatingSystem method*), 115
should_match() (*robot.libraries.BuiltIn.BuiltIn method*), 95
should_match_regexp() (*robot.libraries.BuiltIn.BuiltIn method*), 95
should_not_be_empty() (*robot.libraries.BuiltIn.BuiltIn method*), 95
should_not_be_equal() (*robot.libraries.BuiltIn.BuiltIn method*), 96
should_not_be_equal_as_integers() (*robot.libraries.BuiltIn.BuiltIn method*), 96
should_not_be_equal_as_numbers() (*robot.libraries.BuiltIn.BuiltIn method*), 96
should_not_be_equal_as_strings() (*robot.libraries.BuiltIn.BuiltIn method*), 96
should_not_be_string() (*robot.libraries.String.String method*), 136
should_not_be_true() (*robot.libraries.BuiltIn.BuiltIn method*), 96
should_not_contain() (*robot.libraries.BuiltIn.BuiltIn method*), 97
should_not_contain_any() (*robot.libraries.BuiltIn.BuiltIn method*), 97
should_not_contain_match() (*robot.libraries.Collections.Collections method*), 101
should_not_end_with() (*robot.libraries.BuiltIn.BuiltIn method*), 97
should_not_exist() (*robot.libraries.OperatingSystem.OperatingSystem method*), 115
should_not_match() (*robot.libraries.BuiltIn.BuiltIn method*), 97
should_not_match_regexp() (*robot.libraries.BuiltIn.BuiltIn method*), 97
should_not_start_with() (*robot.libraries.BuiltIn.BuiltIn method*), 97
should_start_with() (*robot.libraries.BuiltIn.BuiltIn method*), 98

show() (*robot.libdocpkg.consoleviewer.ConsoleViewer method*), 70
show() (*robot.libraries.dialogs_py.InputDialog method*), 179
show() (*robot.libraries.dialogs_py.MessageDialog method*), 165
show() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 207
show() (*robot.libraries.dialogs_py.PassFailDialog method*), 221
show() (*robot.libraries.dialogs_py.SelectionDialog method*), 193
single_request() (*robot.libraries.Remote.TimeoutHTTPTransport method*), 130
single_request() (*robot.libraries.Remote.TimeoutHTTPTransport method*), 129
single_value (*robot.parsing.lexer.settings.InitFileSettings attribute*), 326
single_value (*robot.parsing.lexer.settings.KeywordSettings attribute*), 327
single_value (*robot.parsing.lexer.settings.ResourceFileSettings attribute*), 327
single_value (*robot.parsing.lexer.settings.Settings attribute*), 326
single_value (*robot.parsing.lexer.settings.TestCaseFileSettings attribute*), 326
single_value (*robot.parsing.lexer.settings.TestCaseSettings attribute*), 327
SingleTagPattern (*class in robot.model.tags*), 277
SingleType (*class in robot.parsing.lexer.statementlexers*), 328
SingleValue (*class in robot.parsing.model.statements*), 347
size() (*robot.libraries.dialogs_py.InputDialog method*), 179
size() (*robot.libraries.dialogs_py.MessageDialog method*), 165
size() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 207
size() (*robot.libraries.dialogs_py.PassFailDialog method*), 221
size() (*robot.libraries.dialogs_py.SelectionDialog method*), 193
sizefrom() (*robot.libraries.dialogs_py.InputDialog method*), 179
sizefrom() (*robot.libraries.dialogs_py.MessageDialog method*), 165
sizefrom() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 207
sizefrom() (*robot.libraries.dialogs_py.PassFailDialog method*), 221
sizefrom() (*robot.libraries.dialogs_py.SelectionDialog method*), 193
skip (*robot.conf.settings.RobotSettings attribute*), 65
SKIP (*robot.result.model.Break attribute*), 488
SKIP (*robot.result.model.Continue attribute*), 486
SKIP (*robot.result.model.For attribute*), 468
SKIP (*robot.result.model.ForIteration attribute*), 465
SKIP (*robot.result.model.If attribute*), 477
SKIP (*robot.result.model.IfBranch attribute*), 474
SKIP (*robot.result.model.Keyword attribute*), 491
SKIP (*robot.result.model.Return attribute*), 484
SKIP (*robot.result.model.StatusMixin attribute*), 464
SKIP (*robot.result.model.TestCase attribute*), 494
SKIP (*robot.result.model.TestSuite attribute*), 496
SKIP (*robot.result.model.Try attribute*), 481
SKIP (*robot.result.model.TryBranch attribute*), 479
SKIP (*robot.result.model.While attribute*), 472
SKIP (*robot.result.model.WhileIteration attribute*), 470
skip() (*robot.libraries.BuiltIn.BuiltIn method*), 98
skip() (*robot.output.filelogger.FileLogger method*), 305
skip() (*robot.output.logger.Logger method*), 309
skip() (*robot.output.loggerhelper.AbstractLogger method*), 309
skip() (*robot.output.output.Output method*), 311
skip_if() (*robot.libraries.BuiltIn.BuiltIn method*), 98
skip_on_failure (*robot.conf.settings.RobotSettings attribute*), 65
skip_on_failure_after_tag_changes (*robot.running.status.TestStatus attribute*), 583
skip_teardown_on_exit (*robot.conf.settings.RobotSettings attribute*), 65
SkipExecution, 13
skipped (*robot.model.stats.Stat attribute*), 275
skipped (*robot.model.totalstatistics.TotalStatistics attribute*), 288
skipped (*robot.result.model.Break attribute*), 489
skipped (*robot.result.model.Continue attribute*), 487
skipped (*robot.result.model.For attribute*), 469
skipped (*robot.result.model.ForIteration attribute*), 466
skipped (*robot.result.model.If attribute*), 478
skipped (*robot.result.model.IfBranch attribute*), 476
skipped (*robot.result.model.Keyword attribute*), 492
skipped (*robot.result.model.Return attribute*), 485
skipped (*robot.result.model.StatusMixin attribute*), 464
skipped (*robot.result.model.TestCase attribute*), 495
skipped (*robot.result.model.TestSuite attribute*), 496
skipped (*robot.result.model.Try attribute*), 483
skipped (*robot.result.model.TryBranch attribute*), 480
skipped (*robot.result.model.While attribute*), 473
skipped (*robot.result.model.WhileIteration attribute*), 471
skipped_tags (*robot.conf.settings.RobotSettings attribute*), 65

slaves() (*robot.libraries.dialogs_py.InputDialog method*), 179
slaves() (*robot.libraries.dialogs_py.MessageDialog method*), 165
slaves() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 207
slaves() (*robot.libraries.dialogs_py.PassFailDialog method*), 221
slaves() (*robot.libraries.dialogs_py.SelectionDialog method*), 193
sleep() (*robot.libraries.BuiltIn.BuiltIn method*), 98
sock_avail() (*robot.libraries.Telnet.TelnetConnection method*), 146
sort() (*robot.model.body.BaseBody method*), 230
sort() (*robot.model.body.Body method*), 232
sort() (*robot.model.body.Branches method*), 234
sort() (*robot.model.itemlist.ItemList method*), 259
sort() (*robot.model.keyword.Keywords method*), 262
sort() (*robot.model.message.Messages method*), 264
sort() (*robot.model.testcase.TestCases method*), 285
sort() (*robot.model.testsuite.TestSuites method*), 288
sort() (*robot.result.model.Body method*), 459
sort() (*robot.result.model.Branches method*), 461
sort() (*robot.result.model.Iterations method*), 462
sort() (*robot.running.model.Body method*), 555
sort() (*robot.running.model.Imports method*), 577
sort_list() (*robot.libraries.Collections.Collections method*), 106
source (*robot.model.testcase.TestCase attribute*), 284
source (*robot.model.testsuite.TestSuite attribute*), 285
source (*robot.result.executionresult.Result attribute*), 413
source (*robot.result.model.TestCase attribute*), 495
source (*robot.result.model.TestSuite attribute*), 498
source (*robot.running.model.Break attribute*), 569
source (*robot.running.model.Continue attribute*), 567
source (*robot.running.model.For attribute*), 557
source (*robot.running.model.If attribute*), 562
source (*robot.running.model.IfBranch attribute*), 560
source (*robot.running.model.Keyword attribute*), 555
source (*robot.running.model.Return attribute*), 566
source (*robot.running.model.TestCase attribute*), 570
source (*robot.running.model.TestSuite attribute*), 575
source (*robot.running.model.Try attribute*), 564
source (*robot.running.model.TryBranch attribute*), 563
source (*robot.running.model.UserKeyword attribute*), 576
source (*robot.running.model.While attribute*), 559
source (*robot.running.userkeywordrunner.EmbeddedArgumentsRunner attribute*), 590
source (*robot.running.userkeywordrunner.UserKeywordRunner attribute*), 590
sourcename (*robot.result.model.Keyword attribute*), 490
split_command_line() (*robot.libraries.Process.Process method*), 127
split_extension() (*robot.libraries.OperatingSystem.OperatingSystem method*), 120
split_from_equals() (*in module robot.utils.escaping*), 598
split_log (*robot.conf.settings.RebotSettings attribute*), 66
split_log (*robot.conf.settings.RobotSettings attribute*), 66
split_path() (*robot.libraries.OperatingSystem.OperatingSystem method*), 120
split_string() (*robot.libraries.String.String method*), 135
split_string_from_right() (*robot.libraries.String.String method*), 136
split_string_to_characters() (*robot.libraries.String.String method*), 136
split_to_lines() (*robot.libraries.String.String method*), 133
SplitLogWriter (*class in robot.reporting.jswriter*), 395
STANDARD (*robot.libdocpkg.datatypes.TypeDoc attribute*), 70
start() (*robot.result.xmlelementhandlers.ArgumentHandler method*), 526
start() (*robot.result.xmlelementhandlers.ArgumentsHandler method*), 525
start() (*robot.result.xmlelementhandlers.AssignHandler method*), 525
start() (*robot.result.xmlelementhandlers.BranchHandler method*), 521
start() (*robot.result.xmlelementhandlers.BreakHandler method*), 522
start() (*robot.result.xmlelementhandlers.ContinueHandler method*), 522
start() (*robot.result.xmlelementhandlers.DocHandler method*), 523
start() (*robot.result.xmlelementhandlers.ElementHandler method*), 519
start() (*robot.result.xmlelementhandlers.ErrorMessageHandler method*), 526
start() (*robot.result.xmlelementhandlers.ErrorsHandler method*), 526
start() (*robot.result.xmlelementhandlers.ForHandler method*), 520
start() (*robot.result.xmlelementhandlers.IfHandler method*), 521
start() (*robot.result.xmlelementhandlers.IterationHandler method*), 521
start() (*robot.result.xmlelementhandlers.KeywordHandler method*), 520

start() (*robot.result.xmlelementhandlers.MessageHandler method*), 523
start() (*robot.result.xmlelementhandlers.MetadataHandler method*), 523
start() (*robot.result.xmlelementhandlers.MetadataItemHandler method*), 524
start() (*robot.result.xmlelementhandlers.MetaHandler method*), 524
start() (*robot.result.xmlelementhandlers.PatternHandler method*), 522
start() (*robot.result.xmlelementhandlers.ReturnHandler method*), 522
start() (*robot.result.xmlelementhandlers.RobotHandler method*), 519
start() (*robot.result.xmlelementhandlers.RootHandler method*), 519
start() (*robot.result.xmlelementhandlers.StatisticsHandler method*), 527
start() (*robot.result.xmlelementhandlers.StatusHandler method*), 523
start() (*robot.result.xmlelementhandlers.SuiteHandler method*), 519
start() (*robot.result.xmlelementhandlers.TagHandler method*), 524
start() (*robot.result.xmlelementhandlers.TagsHandler method*), 524
start() (*robot.result.xmlelementhandlers.TestHandler method*), 520
start() (*robot.result.xmlelementhandlers.TimeoutHandler method*), 525
start() (*robot.result.xmlelementhandlers.TryHandler method*), 521
start() (*robot.result.xmlelementhandlers.ValueHandler method*), 526
start() (*robot.result.xmlelementhandlers.VarHandler method*), 525
start() (*robot.result.xmlelementhandlers.WhileHandler method*), 520
start() (*robot.result.xmlelementhandlers.XmlElementHandler method*), 519
start() (*robot.running.timeouts.KeywordTimeout method*), 548
start() (*robot.running.timeouts.TestTimeout method*), 548
start() (*robot.utils.markupwriters.HtmlWriter method*), 602
start() (*robot.utils.markupwriters.NullMarkupWriter method*), 603
start() (*robot.utils.markupwriters.XmlWriter method*), 603
start_block() (*robot.parsing.model.blocks.ValidationContext method*), 345
start_body_item() (*robot.conf.gatherfailed.GatherFailedSuites method*), 29
start_body_item() (*robot.conf.gatherfailed.GatherFailedTests method*), 24
start_body_item() (*robot.model.configurer.SuiteConfigurer method*), 235
start_body_item() (*robot.model.filter.EmptySuiteRemover method*), 251
start_body_item() (*robot.model.filter.Filter method*), 256
start_body_item() (*robot.model.modifier.ModelModifier method*), 267
start_body_item() (*robot.model.statistics.StatisticsBuilder method*), 272
start_body_item() (*robot.model.tagsetter.TagSetter method*), 279
start_body_item() (*robot.model.totalstatistics.TotalStatisticsBuilder method*), 290
start_body_item() (*robot.model.visitor.SuiteVisitor method*), 298
start_body_item() (*robot.output.console.dotted.StatusReporter method*), 300
start_body_item() (*robot.output.xmllogger.XmlLogger method*), 316
start_body_item() (*robot.reporting.outputwriter.OutputWriter method*), 397
start_body_item() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 404
start_body_item() (*robot.result.configurer.SuiteConfigurer method*), 409
start_body_item() (*robot.result.keywordremover.AllKeywordsRemover method*), 417
start_body_item() (*robot.result.keywordremover.ByNameKeywordRemover method*), 425
start_body_item() (*robot.result.keywordremover.ByTagKeywordRemover method*), 429
start_body_item() (*robot.result.keywordremover.ForLoopItemsRemover method*), 433

start_body_item() (*robot.result.keywordremover.PassedKeywordRemover method*), 421
start_body_item() (*robot.result.keywordremover.WaitUntilKeywordSucceedsRemover method*), 442
start_body_item() (*robot.result.keywordremover.WarningAndErrorFinder method*), 446
start_body_item() (*robot.result.keywordremover.WhileLoopItemsRemover method*), 437
start_body_item() (*robot.result.merger.Merger method*), 450
start_body_item() (*robot.result.messagefilter.MessageFilter method*), 455
start_body_item() (*robot.result.resultbuilder.RemoveKeywords method*), 502
start_body_item() (*robot.result.suiteteardownfailed.SuiteTeardownFailed method*), 511
start_body_item() (*robot.result.suiteteardownfailed.SuiteTeardownFailureHandler method*), 506
start_body_item() (*robot.result.visitor.ResultVisitor method*), 515
start_body_item() (*robot.running.randomizer.Randomizer method*), 580
start_body_item() (*robot.running.suiterunner.SuiteRunner method*), 586
start_break() (*robot.conf.gatherfailed.GatherFailedSuites method*), 29
start_break() (*robot.conf.gatherfailed.GatherFailedTests method*), 25
start_break() (*robot.model.configurer.SuiteConfigurer method*), 235
start_break() (*robot.model.filter.EmptySuiteRemover method*), 252
start_break() (*robot.model.filter.Filter method*), 256
start_break() (*robot.model.modifier.ModelModifier method*), 267
start_break() (*robot.model.statistics.StatisticsBuilder method*), 272
start_break() (*robot.model.tagsetter.TagSetter method*), 279
start_break() (*robot.model.totalstatistics.TotalStatisticsBuilder method*), 290
start_break() (*robot.model.visitor.SuiteVisitor method*), 297
start_break() (*robot.output.console.dotted.StatusReporter method*), 300
start_break() (*robot.output.xmllogger.XmlLogger method*), 315
start_break() (*robot.reporting.outputwriter.OutputWriter method*), 397
start_break() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 404
start_break() (*robot.result.configurer.SuiteConfigurer method*), 410
start_break() (*robot.result.keywordremover.AllKeywordsRemover method*), 417
start_break() (*robot.result.keywordremover.ByNameKeywordRemover method*), 425
start_break() (*robot.result.keywordremover.ByTagKeywordRemover method*), 429
start_break() (*robot.result.keywordremover.ForLoopItemsRemover method*), 433
start_break() (*robot.result.keywordremover.PassedKeywordRemover method*), 421
start_break() (*robot.result.keywordremover.WaitUntilKeywordSucceed method*), 442
start_break() (*robot.result.keywordremover.WarningAndErrorFinder method*), 446
start_break() (*robot.result.keywordremover.WhileLoopItemsRemover method*), 438
start_break() (*robot.result.merger.Merger method*), 450
start_break() (*robot.result.messagefilter.MessageFilter method*), 455
start_break() (*robot.result.resultbuilder.RemoveKeywords method*), 502
start_break() (*robot.result.suiteteardownfailed.SuiteTeardownFailed method*), 511
start_break() (*robot.result.suiteteardownfailed.SuiteTeardownFailure method*), 506
start_break() (*robot.result.visitor.ResultVisitor method*), 516
start_break() (*robot.running.randomizer.Randomizer method*), 580
start_break() (*robot.running.suiterunner.SuiteRunner method*), 586
start_continue() (*robot.conf.gatherfailed.GatherFailedSuites method*), 29
start_continue() (*robot.conf.gatherfailed.GatherFailedTests method*), 25
start_continue() (*robot.model.configurer.SuiteConfigurer method*), 236
start_continue() (*robot.model.filter.EmptySuiteRemover method*), 252
start_continue() (*robot.model.filter.Filter method*), 256
start_continue() (*robot.model.modifier.ModelModifier*

method), 267
start_continue() (*robot.model.statistics.StatisticsBuilder method*), 272
start_continue() (*robot.model.tagsetter.TagSetter method*), 279
start_continue() (*robot.model.totalstatistics.TotalStatisticsBuilder method*), 290
start_continue() (*robot.model.visitor.SuiteVisitor method*), 297
start_continue() (*robot.output.console.dotted.StatusReporter method*), 300
start_continue() (*robot.output.xmllogger.XmlLogger method*), 315
start_continue() (*robot.reporting.outputwriter.OutputWriter method*), 397
start_continue() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 404
start_continue() (*robot.result.configurer.SuiteConfigurer method*), 410
start_continue() (*robot.result.keywordremover.AllKeywordsRemover method*), 417
start_continue() (*robot.result.keywordremover.ByNameKeywordRemover method*), 425
start_continue() (*robot.result.keywordremover.ByTagKeywordRemover method*), 429
start_continue() (*robot.result.keywordremover.ForLoopItemsRemover method*), 433
start_continue() (*robot.result.keywordremover.PassedKeywordRemover method*), 421
start_continue() (*robot.result.keywordremover.WaitUntilKeywordSucceedsRemover method*), 442
start_continue() (*robot.result.keywordremover.WarningAndErrorFinder method*), 446
start_continue() (*robot.result.keywordremover.WhileLoopItemsRemover method*), 438
start_continue() (*robot.result.merger.Merger method*), 451
start_continue() (*robot.result.messagefilter.MessageFilter method*), 455
start_continue() (*robot.result.resultbuilder.RemoveKeywords method*), 502
start_continue() (*robot.result.suiteteardownfailed.SuiteTeardownFailed method*), 511
start_continue() (*robot.result.suiteteardownfailed.SuiteTeardownFailureHandler method*), 507
start_continue() (*robot.result.visitor.ResultVisitor method*), 516
start_continue() (*robot.running.randomizer.Randomizer method*), 580
start_continue() (*robot.running.suiterunner.SuiteRunner method*), 586
start_directory() (*robot.parsing.suitestructure.SuiteStructureVisitor method*), 393
start_directory() (*robot.running.builder.builders.SuiteStructureParser method*), 542
start_errors() (*robot.output.xmllogger.XmlLogger method*), 316
start_errors() (*robot.reporting.outputwriter.OutputWriter method*), 397
start_errors() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 404
start_errors() (*robot.result.visitor.ResultVisitor method*), 514
start_for() (*robot.conf.gatherfailed.GatherFailedSuites method*), 29
start_for() (*robot.conf.gatherfailed.GatherFailedTests method*), 25
start_for() (*robot.model.configurer.SuiteConfigurer method*), 236
start_for() (*robot.model.filter.EmptySuiteRemover method*), 252
start_for() (*robot.model.filter.Filter method*), 256
start_for() (*robot.model.modifier.ModelModifier method*), 267
start_for() (*robot.model.statistics.StatisticsBuilder method*), 272
start_for() (*robot.model.tagsetter.TagSetter method*), 279
start_for() (*robot.model.totalstatistics.TotalStatisticsBuilder method*), 290
start_for() (*robot.model.visitor.SuiteVisitor method*), 298
start_for() (*robot.output.console.dotted.StatusReporter method*), 300
start_for() (*robot.output.xmllogger.XmlLogger method*), 314
start_for() (*robot.reporting.outputwriter.OutputWriter method*), 397
start_for() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 404
start_for() (*robot.result.configurer.SuiteConfigurer method*), 410
start_for() (*robot.result.keywordremover.AllKeywordsRemover method*), 417
start_for() (*robot.result.keywordremover.ByNameKeywordRemover method*), 425
start_for() (*robot.result.keywordremover.ByTagKeywordRemover method*), 429
start_for() (*robot.result.keywordremover.ForLoopItemsRemover method*), 432
start_for() (*robot.result.keywordremover.PassedKeywordRemover method*), 421
start_for() (*robot.result.keywordremover.WaitUntilKeywordSucceedsR method*), 442
start_for() (*robot.result.keywordremover.WarningAndErrorFinder method*), 446

start_for() (*robot.result.keywordremover.WhileLoopItemsRemover method*), 438
start_for() (*robot.result.merger.Merger method*), 451
start_for() (*robot.result.messagefilter.MessageFilter method*), 455
start_for() (*robot.result.resultbuilder.RemoveKeywords method*), 502
start_for() (*robot.result.suiteteardownfailed.SuiteTeardownFailed method*), 511
start_for() (*robot.result.suiteteardownfailed.SuiteTeardownFailureHandler method*), 507
start_for() (*robot.result.visitor.ResultVisitor method*), 516
start_for() (*robot.running.randomizer.Randomizer method*), 580
start_for() (*robot.running.suiterunner.SuiteRunner method*), 586
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*), 236
start_for_iteration() (*robot.model.filter.EmptySuiteRemover method*), 252
start_for_iteration() (*robot.model.filter.Filter method*), 256
start_for_iteration() (*robot.model.modifier.ModelModifier method*), 267
start_for_iteration() (*robot.model.statistics.StatisticsBuilder method*), 272
start_for_iteration() (*robot.model.tagsetter.TagSetter method*), 279
start_for_iteration() (*robot.model.totalstatistics.TotalStatisticsBuilder method*), 290
start_for_iteration() (*robot.model.visitor.SuiteVisitor method*), 295
start_for_iteration() (*robot.output.console.dotted.StatusReporter method*), 300
start_for_iteration() (*robot.output.xmllogger.XmlLogger method*), 314
start_for_iteration() (*robot.reporting.outputwriter.OutputWriter method*), 398
start_for_iteration() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 404
start_for_iteration() (*robot.result.configurer.SuiteConfigurer method*), 410
start_for_iteration() (*robot.result.keywordremover.AllKeywordsRemover method*), 417
start_for_iteration() (*robot.result.keywordremover.ByNameKeywordRemover method*), 425
start_for_iteration() (*robot.result.keywordremover.ByTagKeywordRemover method*), 430
start_for_iteration() (*robot.result.keywordremover.ForLoopItemsRemover method*), 434
start_for_iteration() (*robot.result.keywordremover.PassedKeywordRemover method*), 421
start_for_iteration() (*robot.result.keywordremover.WaitUntilKeywordSucceedsRemover method*), 442
start_for_iteration() (*robot.result.keywordremover.WarningAndErrorFinder method*), 447
start_for_iteration() (*robot.result.keywordremover.WhileLoopItemsRemover method*), 438
start_for_iteration() (*robot.result.merger.Merger method*), 451
start_for_iteration() (*robot.result.messagefilter.MessageFilter method*), 455
start_for_iteration() (*robot.result.resultbuilder.RemoveKeywords method*), 502
start_for_iteration() (*robot.result.suiteteardownfailed.SuiteTeardownFailed method*), 511
start_for_iteration() (*robot.result.suiteteardownfailed.SuiteTeardownFailureHandler method*), 507
start_for_iteration() (*robot.result.visitor.ResultVisitor method*), 516
start_for_iteration() (*robot.running.randomizer.Randomizer method*), 580
start_for_iteration() (*robot.running.suiterunner.SuiteRunner*

method), 586
start_if() (*robot.conf.gatherfailed.GatherFailedSuites method*), 29
start_if() (*robot.conf.gatherfailed.GatherFailedTests method*), 25
start_if() (*robot.model.configurer.SuiteConfigurer method*), 236
start_if() (*robot.model.filter.EmptySuiteRemover method*), 252
start_if() (*robot.model.filter.Filter method*), 256
start_if() (*robot.model.modifier.ModelModifier method*), 267
start_if() (*robot.model.statistics.StatisticsBuilder method*), 272
start_if() (*robot.model.tagsetter.TagSetter method*), 279
start_if() (*robot.model.totalstatistics.TotalStatisticsBuilder method*), 290
start_if() (*robot.model.visitor.SuiteVisitor method*), 295
start_if() (*robot.output.console.dotted.StatusReporter method*), 301
start_if() (*robot.output.xmllogger.XmlLogger method*), 313
start_if() (*robot.reporting.outputwriter.OutputWriter method*), 398
start_if() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 404
start_if() (*robot.result.configurer.SuiteConfigurer method*), 410
start_if() (*robot.result.keywordremover.AllKeywordsRemover method*), 417
start_if() (*robot.result.keywordremover.ByNameKeywordRemover method*), 425
start_if() (*robot.result.keywordremover.ByTagKeywordRemover method*), 430
start_if() (*robot.result.keywordremover.ForLoopItemsRemover method*), 434
start_if() (*robot.result.keywordremover.PassedKeywordRemover method*), 421
start_if() (*robot.result.keywordremover.WaitUntilKeywordSucceedsRemover method*), 442
start_if() (*robot.result.keywordremover.WarningAndErrorFinder method*), 447
start_if() (*robot.result.keywordremover.WhileLoopItemsRemover method*), 438
start_if() (*robot.result.merger.Merger method*), 451
start_if() (*robot.result.messagefilter.MessageFilter method*), 455
start_if() (*robot.result.resultbuilder.RemoveKeywords method*), 503
start_if() (*robot.result.suiteteardownfailed.SuiteTeardownFailed method*), 511
start_if() (*robot.result.suiteteardownfailed.SuiteTeardownFailureHandler method*), 507
start_if() (*robot.result.visitor.ResultVisitor method*), 516
start_if() (*robot.running.randomizer.Randomizer method*), 580
start_if() (*robot.running.suiterunner.SuiteRunner method*), 586
start_if_branch() (*robot.conf.gatherfailed.GatherFailedSuites method*), 29
start_if_branch() (*robot.conf.gatherfailed.GatherFailedTests method*), 25
start_if_branch() (*robot.model.configurer.SuiteConfigurer method*), 236
start_if_branch() (*robot.model.filter.EmptySuiteRemover method*), 252
start_if_branch() (*robot.model.filter.Filter method*), 256
start_if_branch() (*robot.model.modifier.ModelModifier method*), 267
start_if_branch() (*robot.model.statistics.StatisticsBuilder method*), 272
start_if_branch() (*robot.model.tagsetter.TagSetter method*), 279
start_if_branch() (*robot.model.totalstatistics.TotalStatisticsBuilder method*), 290
start_if_branch() (*robot.model.visitor.SuiteVisitor method*), 296
start_if_branch() (*robot.output.console.dotted.StatusReporter method*), 301
start_if_branch() (*robot.output.xmllogger.XmlLogger method*), 314
start_if_branch() (*robot.reporting.outputwriter.OutputWriter method*), 398
start_if_branch() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 405
start_if_branch() (*robot.result.configurer.SuiteConfigurer method*), 410
start_if_branch() (*robot.result.keywordremover.AllKeywordsRemover method*), 417

start_if_branch() (*robot.result.keywordremover.ByNameKeywordRemover method*), 426
start_if_branch() (*robot.result.keywordremover.ByTagKeywordRemover method*), 430
start_if_branch() (*robot.result.keywordremover.ForLoopItemsRemover method*), 434
start_if_branch() (*robot.result.keywordremover.PassedKeywordRemover method*), 422
start_if_branch() (*robot.result.keywordremover.WaitUntilKeywordSucceedsRemover method*), 442
start_if_branch() (*robot.result.keywordremover.WarningAndErrorFinder method*), 447
start_if_branch() (*robot.result.keywordremover.WhileLoopItemsRemover method*), 438
start_if_branch() (*robot.result.merger.Merger method*), 451
start_if_branch() (*robot.result.messagefilter.MessageFilter method*), 455
start_if_branch() (*robot.result.resultbuilder.RemoveKeywords method*), 503
start_if_branch() (*robot.result.suiteteardownfailed.SuiteTeardownFailed method*), 511
start_if_branch() (*robot.result.suiteteardownfailed.SuiteTeardownFailureHandler method*), 507
start_if_branch() (*robot.result.visitor.ResultVisitor method*), 516
start_if_branch() (*robot.running.randomizer.Randomizer method*), 580
start_if_branch() (*robot.running.suiterunner.SuiteRunner method*), 587
start_keyword() (*robot.conf.gatherfailed.GatherFailedSuites method*), 29
start_keyword() (*robot.conf.gatherfailed.GatherFailedTests method*), 25
start_keyword() (*robot.model.configurer.SuiteConfigurer method*), 236
start_keyword() (*robot.model.filter.EmptySuiteRemover method*), 252
start_keyword() (*robot.model.filter.Filter method*), 256
start_keyword() (*robot.model.modifier.ModelModifier method*), 267
start_keyword() (*robot.model.statistics.StatisticsBuilder method*), 272
start_keyword() (*robot.model.tagsetter.TagSetter method*), 280
start_keyword() (*robot.model.totalstatistics.TotalStatisticsBuilder method*), 291
start_keyword() (*robot.model.visitor.SuiteVisitor method*), 294
start_keyword() (*robot.output.console.dotted.StatusReporter method*), 301
start_keyword() (*robot.output.console.verbose.VerboseOutput method*), 304
start_keyword() (*robot.output.filelogger.FileLogger method*), 305
start_keyword() (*robot.output.listeners.Listeners method*), 307
start_keyword() (*robot.output.logger.Logger method*), 308
start_keyword() (*robot.output.logger.LoggerProxy method*), 309
start_keyword() (*robot.output.output.Output method*), 311
start_keyword() (*robot.output.xmllogger.XmlLogger method*), 313
start_keyword() (*robot.reporting.outputwriter.OutputWriter method*), 398
start_keyword() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 405
start_keyword() (*robot.result.configurer.SuiteConfigurer method*), 410
start_keyword() (*robot.result.keywordremover.AllKeywordsRemover method*), 418
start_keyword() (*robot.result.keywordremover.ByNameKeywordRemo method*), 424
start_keyword() (*robot.result.keywordremover.ByTagKeywordRemove method*), 428
start_keyword() (*robot.result.keywordremover.ForLoopItemsRemover method*), 434
start_keyword() (*robot.result.keywordremover.PassedKeywordRemov method*), 422
start_keyword() (*robot.result.keywordremover.WaitUntilKeywordSucc method*), 440
start_keyword() (*robot.result.keywordremover.WarningAndErrorFind method*), 445
start_keyword() (*robot.result.keywordremover.WhileLoopItemsRemov method*), 438
start_keyword() (*robot.result.merger.Merger method*), 451
start_keyword() (*robot.result.messagefilter.MessageFilter method*), 453
start_keyword() (*robot.result.resultbuilder.RemoveKeywords method*), 503

start_keyword() (robot.result.suiteteardownfailed.SuiteTeardownFailed method), 511
start_keyword() (robot.result.suiteteardownfailed.SuiteTeardownFailureHandler method), 507
start_keyword() (robot.result.visitor.ResultVisitor method), 516
start_keyword() (robot.running.randomizer.Randomizer method), 580
start_keyword() (robot.running.suiterunner.SuiteRunner method), 587
start_keyword() (robot.variables.scopes.SetVariables method), 610
start_keyword() (robot.variables.scopes.VariableScopes method), 609
start_loggers (robot.output.logger.Logger attribute), 308
start_message() (robot.conf.gatherfailed.GatherFailedSuites method), 30
start_message() (robot.conf.gatherfailed.GatherFailedTests method), 25
start_message() (robot.model.configurer.SuiteConfigurer method), 236
start_message() (robot.model.filter.EmptySuiteRemover method), 252
start_message() (robot.model.filter.Filter method), 256
start_message() (robot.model.modifier.ModelModifier method), 267
start_message() (robot.model.statistics.StatisticsBuilder method), 272
start_message() (robot.model.tagsetter.TagSetter method), 280
start_message() (robot.model.totalstatistics.TotalStatisticsBuilder method), 291
start_message() (robot.model.visitor.SuiteVisitor method), 298
start_message() (robot.output.console.dotted.StatusReporter method), 301
start_message() (robot.output.xmllogger.XmlLogger method), 316
start_message() (robot.reporting.outputwriter.OutputWriter method), 395
start_message() (robot.reporting.xunitwriter.XUnitFileWriter method), 405
start_message() (robot.result.configurer.SuiteConfigurer method), 410
start_message() (robot.result.keywordremover.AllKeywordsRemover method), 418
start_message() (robot.result.keywordremover.ByNameKeywordRemover method), 426
start_message() (robot.result.keywordremover.ByTagKeywordRemover method), 430
start_message() (robot.result.keywordremover.ForLoopItemsRemover method), 434
start_message() (robot.result.keywordremover.PassedKeywordRemover method), 422
start_message() (robot.result.keywordremover.WaitUntilKeywordSucceeds method), 442
start_message() (robot.result.keywordremover.WarningAndErrorFinder method), 447
start_message() (robot.result.keywordremover.WhileLoopItemsRemover method), 438
start_message() (robot.result.merger.Merger method), 451
start_message() (robot.result.messagefilter.MessageFilter method), 455
start_message() (robot.result.resultbuilder.RemoveKeywords method), 503
start_message() (robot.result.suiteteardownfailed.SuiteTeardownFailed method), 511
start_message() (robot.result.suiteteardownfailed.SuiteTeardownFailureHandler method), 507
start_message() (robot.result.visitor.ResultVisitor method), 516
start_message() (robot.running.randomizer.Randomizer method), 581
start_message() (robot.running.suiterunner.SuiteRunner method), 587
start_process() (robot.libraries.Process.Process method), 125
start_result() (robot.output.xmllogger.XmlLogger method), 316
start_result() (robot.reporting.outputwriter.OutputWriter method), 398
start_result() (robot.reporting.xunitwriter.XUnitFileWriter method), 405
start_result() (robot.result.visitor.ResultVisitor method), 514
start_return() (robot.conf.gatherfailed.GatherFailedSuites method), 30
start_return() (robot.conf.gatherfailed.GatherFailedTests method), 25
start_return() (robot.model.configurer.SuiteConfigurer method), 236
start_return() (robot.model.filter.EmptySuiteRemover method), 252
start_return() (robot.model.filter.Filter method), 256
start_return() (robot.model.modifier.ModelModifier method), 268
start_return() (robot.model.statistics.StatisticsBuilder method), 273
start_return() (robot.model.tagsetter.TagSetter method), 280
start_return() (robot.model.totalstatistics.TotalStatisticsBuilder method), 291
start_return() (robot.model.visitor.SuiteVisitor method), 297

start_return() (*robot.output.console.dotted.StatusReporter method*), 301
start_return() (*robot.output.xmllogger.XmlLogger method*), 315
start_return() (*robot.reporting.outputwriter.OutputWriter method*), 398
start_return() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 405
start_return() (*robot.result.configurer.SuiteConfigurer method*), 410
start_return() (*robot.result.keywordremover.AllKeywordsRemover method*), 418
start_return() (*robot.result.keywordremover.ByNameKeywordRemover method*), 426
start_return() (*robot.result.keywordremover.ByTagKeywordRemover method*), 430
start_return() (*robot.result.keywordremover.ForLoopItemsRemover method*), 434
start_return() (*robot.result.keywordremover.PassedKeywordRemover method*), 422
start_return() (*robot.result.keywordremover.WaitUntilKeywordSucceedsRemover method*), 443
start_return() (*robot.result.keywordremover.WarningAndErrorFinder method*), 447
start_return() (*robot.result.keywordremover.WhileLoopItemsRemover method*), 438
start_return() (*robot.result.merger.Merger method*), 451
start_return() (*robot.result.messagefilter.MessageFilter method*), 455
start_return() (*robot.result.resultbuilder.RemoveKeywords method*), 503
start_return() (*robot.result.suiteteardownfailed.SuiteTeardownFailed method*), 511
start_return() (*robot.result.suiteteardownfailed.SuiteTeardownFailureHandler method*), 507
start_return() (*robot.result.visitor.ResultVisitor method*), 516
start_return() (*robot.running.randomizer.Randomizer method*), 581
start_return() (*robot.running.suiterunner.SuiteRunner method*), 587
start_splitting_if_needed() (*robot.reporting.jsbuildingcontext.JsBuildingContext method*), 394
start_stat() (*robot.output.xmllogger.XmlLogger method*), 316
start_stat() (*robot.reporting.outputwriter.OutputWriter method*), 398
start_stat() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 405
start_stat() (*robot.result.visitor.ResultVisitor method*), 514
start_statistics() (*robot.output.xmllogger.XmlLogger method*), 316
start_statistics() (*robot.reporting.outputwriter.OutputWriter method*), 398
start_statistics() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 405
start_statistics() (*robot.result.visitor.ResultVisitor method*), 514
start_suite() (*robot.conf.gatherfailed.GatherFailedSuites method*), 27
start_suite() (*robot.conf.gatherfailed.GatherFailedTests method*), 25
start_suite() (*robot.model.configurer.SuiteConfigurer method*), 236
start_suite() (*robot.model.filter.EmptySuiteRemover method*), 252
start_suite() (*robot.model.filter.Filter method*), 254
start_suite() (*robot.model.modifier.ModelModifier method*), 268
start_suite() (*robot.model.statistics.StatisticsBuilder method*), 270
start_suite() (*robot.model.suitestatistics.SuiteStatisticsBuilder method*), 276
start_suite() (*robot.model.tagsetter.TagSetter method*), 277
start_suite() (*robot.model.totalstatistics.TotalStatisticsBuilder method*), 291
start_suite() (*robot.model.visitor.SuiteVisitor method*), 294
start_suite() (*robot.output.console.dotted.DottedOutput method*), 298
start_suite() (*robot.output.console.dotted.StatusReporter method*), 301
start_suite() (*robot.output.console.verbose.VerboseOutput method*), 304
start_suite() (*robot.output.filelogger.FileLogger method*), 305
start_suite() (*robot.output.logger.Logger method*), 308
start_suite() (*robot.output.output.Output method*), 311
start_suite() (*robot.output.xmllogger.XmlLogger method*), 315
start_suite() (*robot.reporting.outputwriter.OutputWriter method*), 398
start_suite() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 402
start_suite() (*robot.result.configurer.SuiteConfigurer method*), 411
start_suite() (*robot.result.keywordremover.AllKeywordsRemover

method), 418
start_suite() (*robot.result.keywordremover.ByNameKeywordRemover method*), 426
start_suite() (*robot.result.keywordremover.ByTagKeywordRemover method*), 430
start_suite() (*robot.result.keywordremover.ForLoopItemsRemover method*), 434
start_suite() (*robot.result.keywordremover.PassedKeywordRemover method*), 419
start_suite() (*robot.result.keywordremover.WaitUntilKeywordSucceedsRemover method*), 443
start_suite() (*robot.result.keywordremover.WarningAndErrorFinder method*), 445
start_suite() (*robot.result.keywordremover.WhileLoopItemsRemover method*), 439
start_suite() (*robot.result.merger.Merger method*), 449
start_suite() (*robot.result.messagefilter.MessageFilter method*), 453
start_suite() (*robot.result.resultbuilder.RemoveKeywords method*), 501
start_suite() (*robot.result.suiteteardownfailed.SuiteTeardownFailed method*), 512
start_suite() (*robot.result.suiteteardownfailed.SuiteTeardownFailureHandler method*), 507
start_suite() (*robot.result.visitor.ResultVisitor method*), 517
start_suite() (*robot.running.context.ExecutionContexts method*), 550
start_suite() (*robot.running.libraryscopes.GlobalScope method*), 553
start_suite() (*robot.running.libraryscopes.TestCaseScope method*), 553
start_suite() (*robot.running.libraryscopes.TestSuiteScope method*), 553
start_suite() (*robot.running.namespace.Namespace method*), 577
start_suite() (*robot.running.randomizer.Randomizer method*), 578
start_suite() (*robot.running.suiterunner.SuiteRunner method*), 585
start_suite() (*robot.variables.scopes.SetVariables method*), 610
start_suite() (*robot.variables.scopes.VariableScopes method*), 609
start_suite_statistics() (*robot.output.xmllogger.XmlLogger method*), 316
start_suite_statistics() (*robot.reporting.outputwriter.OutputWriter method*), 398
start_suite_statistics() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 405
start_suite_statistics() (*robot.result.visitor.ResultVisitor method*), 514
start_suite_statistics() (*robot.output.xmllogger.XmlLogger method*), 316
start_tag_statistics() (*robot.reporting.outputwriter.OutputWriter method*), 398
start_tag_statistics() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 405
start_tag_statistics() (*robot.result.visitor.ResultVisitor method*), 514
start_test() (*robot.conf.gatherfailed.GatherFailedSuites method*), 30
start_test() (*robot.conf.gatherfailed.GatherFailedTests method*), 26
start_test() (*robot.model.configurer.SuiteConfigurer method*), 236
start_test() (*robot.model.filter.EmptySuiteRemover method*), 253
start_test() (*robot.model.filter.Filter method*), 257
start_test() (*robot.model.modifier.ModelModifier method*), 268
start_test() (*robot.model.statistics.StatisticsBuilder method*), 273
start_test() (*robot.model.tagsetter.TagSetter method*), 280
start_test() (*robot.model.totalstatistics.TotalStatisticsBuilder method*), 291
start_test() (*robot.model.visitor.SuiteVisitor method*), 294
start_test() (*robot.output.console.dotted.StatusReporter method*), 301
start_test() (*robot.output.console.verbose.VerboseOutput method*), 304
start_test() (*robot.output.filelogger.FileLogger method*), 305
start_test() (*robot.output.logger.Logger method*), 308
start_test() (*robot.output.output.Output method*), 311
start_test() (*robot.output.xmllogger.XmlLogger method*), 315
start_test() (*robot.reporting.outputwriter.OutputWriter method*), 398
start_test() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 405
start_test() (*robot.result.configurer.SuiteConfigurer method*), 411
start_test() (*robot.result.keywordremover.AllKeywordsRemover method*), 418

start_test() (*robot.result.keywordremover.ByNameKeywordRemover method*), 426
start_test() (*robot.result.keywordremover.ByTagKeywordRemover method*), 430
start_test() (*robot.result.keywordremover.ForLoopItemsRemover method*), 434
start_test() (*robot.result.keywordremover.PassedKeywordRemover method*), 422
start_test() (*robot.result.keywordremover.WaitUntilKeywordSucceedsRemover method*), 443
start_test() (*robot.result.keywordremover.WarningAndErrorFinder method*), 445
start_test() (*robot.result.keywordremover.WhileLoopItemsRemover method*), 439
start_test() (*robot.result.merger.Merger method*), 451
start_test() (*robot.result.messagefilter.MessageFilter method*), 455
start_test() (*robot.result.resultbuilder.RemoveKeywords method*), 503
start_test() (*robot.result.suiteteardownfailed.SuiteTeardownFailed method*), 512
start_test() (*robot.result.suiteteardownfailed.SuiteTeardownFailureHandler method*), 507
start_test() (*robot.result.visitor.ResultVisitor method*), 517
start_test() (*robot.running.libraryscopes.GlobalScope method*), 553
start_test() (*robot.running.libraryscopes.TestCaseScope method*), 553
start_test() (*robot.running.libraryscopes.TestSuiteScope method*), 553
start_test() (*robot.running.namespace.Namespace method*), 577
start_test() (*robot.running.randomizer.Randomizer method*), 581
start_test() (*robot.running.suiterunner.SuiteRunner method*), 587
start_test() (*robot.variables.scopes.SetVariables method*), 610
start_test() (*robot.variables.scopes.VariableScopes method*), 609
start_total_statistics() (*robot.output.xmllogger.XmlLogger method*), 316
start_total_statistics() (*robot.reporting.outputwriter.OutputWriter method*), 398
start_total_statistics() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 405
start_total_statistics() (*robot.result.visitor.ResultVisitor method*), 514
start_try() (*robot.conf.gatherfailed.GatherFailedSuites method*), 30
start_try() (*robot.conf.gatherfailed.GatherFailedTests method*), 26
start_try() (*robot.model.configurer.SuiteConfigurer method*), 237
start_try() (*robot.model.filter.EmptySuiteRemover method*), 253
start_try() (*robot.model.filter.Filter method*), 257
start_try() (*robot.model.modifier.ModelModifier method*), 268
start_try() (*robot.model.statistics.StatisticsBuilder method*), 273
start_try() (*robot.model.tagsetter.TagSetter method*), 280
start_try() (*robot.model.totalstatistics.TotalStatisticsBuilder method*), 291
start_try() (*robot.model.visitor.SuiteVisitor method*), 296
start_try() (*robot.output.console.dotted.StatusReporter method*), 301
start_try() (*robot.output.xmllogger.XmlLogger method*), 314
start_try() (*robot.reporting.outputwriter.OutputWriter method*), 398
start_try() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 405
start_try() (*robot.result.configurer.SuiteConfigurer method*), 411
start_try() (*robot.result.keywordremover.AllKeywordsRemover method*), 418
start_try() (*robot.result.keywordremover.ByNameKeywordRemover method*), 426
start_try() (*robot.result.keywordremover.ByTagKeywordRemover method*), 430
start_try() (*robot.result.keywordremover.ForLoopItemsRemover method*), 434
start_try() (*robot.result.keywordremover.PassedKeywordRemover method*), 422
start_try() (*robot.result.keywordremover.WaitUntilKeywordSucceedsR method*), 443
start_try() (*robot.result.keywordremover.WarningAndErrorFinder method*), 447
start_try() (*robot.result.keywordremover.WhileLoopItemsRemover method*), 439
start_try() (*robot.result.merger.Merger method*), 451
start_try() (*robot.result.messagefilter.MessageFilter method*), 456
start_try() (*robot.result.resultbuilder.RemoveKeywords method*), 503
start_try() (*robot.result.suiteteardownfailed.SuiteTeardownFailed method*), 512
start_try() (*robot.result.suiteteardownfailed.SuiteTeardownFailureHa*

method), 508
start_try() (*robot.result.visitor.ResultVisitor method*), 517
start_try() (*robot.running.randomizer.Randomizer method*), 581
start_try() (*robot.running.suiterunner.SuiteRunner method*), 587
start_try_branch() (*robot.conf.gatherfailed.GatherFailedSuites method*), 30
start_try_branch() (*robot.conf.gatherfailed.GatherFailedTests method*), 26
start_try_branch() (*robot.model.configurer.SuiteConfigurer method*), 237
start_try_branch() (*robot.model.filter.EmptySuiteRemover method*), 253
start_try_branch() (*robot.model.filter.Filter method*), 257
start_try_branch() (*robot.model.modifier.ModelModifier method*), 268
start_try_branch() (*robot.model.statistics.StatisticsBuilder method*), 273
start_try_branch() (*robot.model.tagsetter.TagSetter method*), 280
start_try_branch() (*robot.model.totalstatistics.TotalStatisticsBuilder method*), 291
start_try_branch() (*robot.model.visitor.SuiteVisitor method*), 296
start_try_branch() (*robot.output.console.dotted.StatusReporter method*), 301
start_try_branch() (*robot.output.xmllogger.XmlLogger method*), 314
start_try_branch() (*robot.reporting.outputwriter.OutputWriter method*), 398
start_try_branch() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 405
start_try_branch() (*robot.result.configurer.SuiteConfigurer method*), 411
start_try_branch() (*robot.result.keywordremover.AllKeywordsRemover method*), 418
start_try_branch() (*robot.result.keywordremover.ByNameKeywordRemover method*), 426
start_try_branch() (*robot.result.keywordremover.ByTagKeywordRemover method*), 430
start_try_branch() (*robot.result.keywordremover.ForLoopItemsRemover method*), 434
start_try_branch() (*robot.result.keywordremover.PassedKeywordRemover method*), 422
start_try_branch() (*robot.result.keywordremover.WaitUntilKeywordSucceedsRemover method*), 443
start_try_branch() (*robot.result.keywordremover.WarningAndErrorFinder method*), 447
start_try_branch() (*robot.result.keywordremover.WhileLoopItemsRemover method*), 439
start_try_branch() (*robot.result.merger.Merger method*), 452
start_try_branch() (*robot.result.messagefilter.MessageFilter method*), 456
start_try_branch() (*robot.result.resultbuilder.RemoveKeywords method*), 503
start_try_branch() (*robot.result.suiteteardownfailed.SuiteTeardownFailed method*), 512
start_try_branch() (*robot.result.suiteteardownfailed.SuiteTeardownFailureHandler method*), 508
start_try_branch() (*robot.result.visitor.ResultVisitor method*), 517
start_try_branch() (*robot.running.randomizer.Randomizer method*), 581
start_try_branch() (*robot.running.suiterunner.SuiteRunner method*), 587
start_user_keyword() (*robot.running.namespace.Namespace method*), 578
start_while() (*robot.conf.gatherfailed.GatherFailedSuites method*), 30
start_while() (*robot.conf.gatherfailed.GatherFailedTests method*), 26
start_while() (*robot.model.configurer.SuiteConfigurer method*), 237
start_while() (*robot.model.filter.EmptySuiteRemover*

method), 253
start_while() (*robot.model.filter.Filter method*), 257
start_while() (*robot.model.modifier.ModelModifier method*), 268
start_while() (*robot.model.statistics.StatisticsBuilder method*), 273
start_while() (*robot.model.tagsetter.TagSetter method*), 280
start_while() (*robot.model.totalstatistics.TotalStatisticsBuilder method*), 291
start_while() (*robot.model.visitor.SuiteVisitor method*), 296
start_while() (*robot.output.console.dotted.StatusReporter method*), 301
start_while() (*robot.output.xmllogger.XmlLogger method*), 314
start_while() (*robot.reporting.outputwriter.OutputWriter method*), 399
start_while() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 405
start_while() (*robot.result.configurer.SuiteConfigurer method*), 411
start_while() (*robot.result.keywordremover.AllKeywordsRemover method*), 418
start_while() (*robot.result.keywordremover.ByNameKeywordRemover method*), 426
start_while() (*robot.result.keywordremover.ByTagKeywordRemover method*), 430
start_while() (*robot.result.keywordremover.ForLoopItemsRemover method*), 435
start_while() (*robot.result.keywordremover.PassedKeywordRemover method*), 422
start_while() (*robot.result.keywordremover.WaitUntilKeywordSucceedsRemover method*), 443
start_while() (*robot.result.keywordremover.WarningAndErrorFinder method*), 447
start_while() (*robot.result.keywordremover.WhileLoopItemsRemover method*), 436
start_while() (*robot.result.merger.Merger method*), 452
start_while() (*robot.result.messagefilter.MessageFilter method*), 456
start_while() (*robot.result.resultbuilder.RemoveKeywords method*), 503
start_while() (*robot.result.suiteteardownfailed.SuiteTeardownFailed method*), 512
start_while() (*robot.result.suiteteardownfailed.SuiteTeardownFailureHandler method*), 508
start_while() (*robot.result.visitor.ResultVisitor method*), 517
start_while() (*robot.running.randomizer.Randomizer method*), 581
start_while() (*robot.running.suiterunner.SuiteRunner method*), 587
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*), 237
start_while_iteration() (*robot.model.filter.EmptySuiteRemover method*), 253
start_while_iteration() (*robot.model.filter.Filter method*), 257
start_while_iteration() (*robot.model.modifier.ModelModifier method*), 268
start_while_iteration() (*robot.model.statistics.StatisticsBuilder method*), 273
start_while_iteration() (*robot.model.tagsetter.TagSetter method*), 280
start_while_iteration() (*robot.model.totalstatistics.TotalStatisticsBuilder method*), 291
start_while_iteration() (*robot.model.visitor.SuiteVisitor method*), 297
start_while_iteration() (*robot.output.console.dotted.StatusReporter method*), 301
start_while_iteration() (*robot.output.xmllogger.XmlLogger method*), 315
start_while_iteration() (*robot.reporting.outputwriter.OutputWriter method*), 399
start_while_iteration() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 405
start_while_iteration() (*robot.result.configurer.SuiteConfigurer method*), 411
start_while_iteration() (*robot.result.keywordremover.AllKeywordsRemover method*), 418
start_while_iteration() (*robot.result.keywordremover.ByNameKeywordRemover method*), 426
start_while_iteration() (*robot.result.keywordremover.ByTagKeywordRemover method*), 430

start_while_iteration() (*robot.result.keywordremover.ForLoopItemsRemover method*), 435
start_while_iteration() (*robot.result.keywordremover.PassedKeywordRemover method*), 422
start_while_iteration() (*robot.result.keywordremover.WaitUntilKeywordSucceedsRemover method*), 443
start_while_iteration() (*robot.result.keywordremover.WarningAndErrorFinder method*), 447
start_while_iteration() (*robot.result.keywordremover.WhileLoopItemsRemover method*), 439
start_while_iteration() (*robot.result.merger.Merger method*), 452
start_while_iteration() (*robot.result.messagefilter.MessageFilter method*), 456
start_while_iteration() (*robot.result.resultbuilder.RemoveKeywords method*), 503
start_while_iteration() (*robot.result.suiteteardownfailed.SuiteTeardownFailed method*), 512
start_while_iteration() (*robot.result.suiteteardownfailed.SuiteTeardownFailureHandler method*), 508
start_while_iteration() (*robot.result.visitor.ResultVisitor method*), 517
start_while_iteration() (*robot.running.randomizer.Randomizer method*), 581
start_while_iteration() (*robot.running.suiterunner.SuiteRunner method*), 587
StartKeywordArguments (*class in robot.output.listenerarguments*), 306
StartSuiteArguments (*class in robot.output.listenerarguments*), 306
StartTestArguments (*class in robot.output.listenerarguments*), 306
starttime (*robot.result.model.Break attribute*), 488
starttime (*robot.result.model.Continue attribute*), 485
starttime (*robot.result.model.For attribute*), 467
starttime (*robot.result.model.ForIteration attribute*), 465
starttime (*robot.result.model.If attribute*), 476
starttime (*robot.result.model.IfBranch attribute*), 474
starttime (*robot.result.model.Keyword attribute*), 490
starttime (*robot.result.model.Return attribute*), 483
starttime (*robot.result.model.TestCase attribute*), 493
starttime (*robot.result.model.TestSuite attribute*), 496
starttime (*robot.result.model.Try attribute*), 481
starttime (*robot.result.model.TryBranch attribute*), 478
starttime (*robot.result.model.While attribute*), 471
starttime (*robot.result.model.WhileIteration attribute*), 469
Stat (*class in robot.model.stats*), 274
stat (*robot.model.suitestatistics.SuiteStatistics attribute*), 276
stat_message (*robot.result.model.TestSuite attribute*), 499
state() (*robot.libraries.dialogs_py.InputDialog method*), 179
state() (*robot.libraries.dialogs_py.MessageDialog method*), 165
state() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 207
state() (*robot.libraries.dialogs_py.PassFailDialog method*), 221
state() (*robot.libraries.dialogs_py.SelectionDialog method*), 193
Statement (*class in robot.parsing.model.statements*), 345
StatementLexer (*class in robot.parsing.lexer.statementlexers*), 327
Statistics (*class in robot.model.statistics*), 270
statistics (*robot.result.executionresult.CombinedResult attribute*), 414
statistics (*robot.result.executionresult.Result attribute*), 413
statistics (*robot.result.model.TestSuite attribute*), 498
statistics_config (*robot.conf.settings.RebotSettings attribute*), 66
statistics_config (*robot.conf.settings.RobotSettings attribute*), 66
StatisticsBuilder (*class in robot.model.statistics*), 270
StatisticsBuilder (*class in robot.reporting.jsmodelbuilders*), 394
StatisticsHandler (*class in robot.result.xmlelementhandlers*), 526
status (*robot.errors.BreakLoop attribute*), 618
status (*robot.errors.ContinueLoop attribute*), 617
status (*robot.errors.ExecutionFailed attribute*), 615
status (*robot.errors.ExecutionFailures attribute*), 616
status (*robot.errors.ExecutionPassed attribute*), 616
status (*robot.errors.ExecutionStatus attribute*), 615
status (*robot.errors.HandlerExecutionFailed attribute*), 615
status (*robot.errors.PassExecution attribute*), 617
status (*robot.errors.ReturnFromKeyword attribute*),

618
status (*robot.errors.UserKeywordExecutionFailed attribute*), 616
status (*robot.result.model.Break attribute*), 487
status (*robot.result.model.Continue attribute*), 485
status (*robot.result.model.For attribute*), 467
status (*robot.result.model.ForIteration attribute*), 465
status (*robot.result.model.If attribute*), 476
status (*robot.result.model.IfBranch attribute*), 474
status (*robot.result.model.Keyword attribute*), 490
status (*robot.result.model.Return attribute*), 483
status (*robot.result.model.TestCase attribute*), 493
status (*robot.result.model.TestSuite attribute*), 498
status (*robot.result.model.Try attribute*), 481
status (*robot.result.model.TryBranch attribute*), 478
status (*robot.result.model.While attribute*), 471
status (*robot.result.model.WhileIteration attribute*), 469
status (*robot.running.status.SuiteStatus attribute*), 583
status (*robot.running.status.TestStatus attribute*), 583
status() (*robot.output.console.verbose.VerboseWriter method*), 304
status_rc (*robot.conf.settings.RebotSettings attribute*), 66
status_rc (*robot.conf.settings.RobotSettings attribute*), 66
StatusHandler (*class in robot.result.xmlelementhandlers*), 523
StatusMixin (*class in robot.result.model*), 464
StatusReporter (*class in robot.output.console.dotted*), 299
StatusReporter (*class in robot.running.statusreporter*), 584
stderr (*robot.libraries.Process.ExecutionResult attribute*), 128
stdout (*robot.libraries.Process.ExecutionResult attribute*), 128
StdoutLogSplitter (*class in robot.output.stdoutlogsplitter*), 313
StoredFinder (*class in robot.variables.finders*), 608
String (*class in robot.libraries.String*), 131
string() (*robot.reporting.jsbuildingcontext.JsBuildingContext method*), 393
StringCache (*class in robot.reporting.stringcache*), 402
StringConverter (*class in robot.running.arguments.typeconverters*), 532
StringDumper (*class in robot.htmldata.jsonwriter*), 68
StringIndex (*class in robot.reporting.stringcache*), 401
strings (*robot.reporting.jsbuildingcontext.JsBuildingContext attribute*), 394
strip() (*robot.libraries.XML.NameSpaceStripper method*), 157
strip_string() (*robot.libraries.String.String method*), 136
subtract_date_from_date() (*in module robot.libraries.DateTime*), 110
subtract_time_from_date() (*in module robot.libraries.DateTime*), 110
subtract_time_from_time() (*in module robot.libraries.DateTime*), 111
suite (*robot.model.statistics.Statistics attribute*), 270
suite (*robot.result.executionresult.Result attribute*), 413
suite_config (*robot.conf.settings.RebotSettings attribute*), 66
suite_config (*robot.conf.settings.RobotSettings attribute*), 65
suite_names (*robot.conf.settings.RebotSettings attribute*), 66
suite_names (*robot.conf.settings.RobotSettings attribute*), 65
suite_separator() (*robot.output.console.verbose.VerboseWriter method*), 304
SUITE_SETUP (*robot.parsing.lexer.tokens.END attribute*), 339
SUITE_SETUP (*robot.parsing.lexer.tokens.EOS attribute*), 337
SUITE_SETUP (*robot.parsing.lexer.tokens.Token attribute*), 334
suite_setup_setting (*robot.conf.languages.Bg attribute*), 59
suite_setup_setting (*robot.conf.languages.Bs attribute*), 38
suite_setup_setting (*robot.conf.languages.Cs attribute*), 36
suite_setup_setting (*robot.conf.languages.De attribute*), 43
suite_setup_setting (*robot.conf.languages.En attribute*), 34
suite_setup_setting (*robot.conf.languages.Es attribute*), 51
suite_setup_setting (*robot.conf.languages.Fi attribute*), 40
suite_setup_setting (*robot.conf.languages.Fr attribute*), 41
suite_setup_setting (*robot.conf.languages.Hi attribute*), 63
suite_setup_setting (*robot.conf.languages.It attribute*), 62
suite_setup_setting (*robot.conf.languages.Language attribute*), 33
suite_setup_setting (*robot.conf.languages.Nl attribute*), 37

suite_setup_setting (*robot.conf.languages.Pl attribute*), 48
suite_setup_setting (*robot.conf.languages.Pt attribute*), 45
suite_setup_setting (*robot.conf.languages.PtBr attribute*), 44
suite_setup_setting (*robot.conf.languages.Ro attribute*), 61
suite_setup_setting (*robot.conf.languages.Ru attribute*), 52
suite_setup_setting (*robot.conf.languages.Sv attribute*), 58
suite_setup_setting (*robot.conf.languages.Th attribute*), 47
suite_setup_setting (*robot.conf.languages.Tr attribute*), 56
suite_setup_setting (*robot.conf.languages.Uk attribute*), 50
suite_setup_setting (*robot.conf.languages.ZhCn attribute*), 54
suite_setup_setting (*robot.conf.languages.ZhTw attribute*), 55
SUITE_TEARDOWN (*robot.parsing.lexer.tokens.END attribute*), 339
SUITE_TEARDOWN (*robot.parsing.lexer.tokens.EOS attribute*), 337
SUITE_TEARDOWN (*robot.parsing.lexer.tokens.Token attribute*), 334
suite_teardown_failed() (*robot.result.model.TestSuite method*), 499
suite_teardown_setting (*robot.conf.languages.Bg attribute*), 59
suite_teardown_setting (*robot.conf.languages.Bs attribute*), 38
suite_teardown_setting (*robot.conf.languages.Cs attribute*), 36
suite_teardown_setting (*robot.conf.languages.De attribute*), 43
suite_teardown_setting (*robot.conf.languages.En attribute*), 34
suite_teardown_setting (*robot.conf.languages.Es attribute*), 51
suite_teardown_setting (*robot.conf.languages.Fi attribute*), 40
suite_teardown_setting (*robot.conf.languages.Fr attribute*), 41
suite_teardown_setting (*robot.conf.languages.Hi attribute*), 64
suite_teardown_setting (*robot.conf.languages.It attribute*), 62
suite_teardown_setting (*robot.conf.languages.Language attribute*), 33
suite_teardown_setting (*robot.conf.languages.Nl attribute*), 37
suite_teardown_setting (*robot.conf.languages.Pl attribute*), 48
suite_teardown_setting (*robot.conf.languages.Pt attribute*), 45
suite_teardown_setting (*robot.conf.languages.PtBr attribute*), 44
suite_teardown_setting (*robot.conf.languages.Ro attribute*), 61
suite_teardown_setting (*robot.conf.languages.Ru attribute*), 52
suite_teardown_setting (*robot.conf.languages.Sv attribute*), 58
suite_teardown_setting (*robot.conf.languages.Th attribute*), 47
suite_teardown_setting (*robot.conf.languages.Tr attribute*), 57
suite_teardown_setting (*robot.conf.languages.Uk attribute*), 50
suite_teardown_setting (*robot.conf.languages.ZhCn attribute*), 54
suite_teardown_setting (*robot.conf.languages.ZhTw attribute*), 55
suite_teardown_skipped() (*robot.result.model.TestSuite method*), 499
SuiteBuilder (*class in robot.reporting.jsmodelbuilders*), 394
SuiteBuilder (*class in robot.running.builder.transformers*), 544
SuiteConfigurer (*class in robot.model.configurer*), 234
SuiteConfigurer (*class in robot.result.configurer*), 408
SuiteDocBuilder (*class in robot.libdocpkg.robotbuilder*), 72
SuiteHandler (*class in robot.result.xmlelementhandlers*), 519
SuiteMessage (*class in robot.running.status*), 584
SuiteNamePatterns (*class in robot.model.namepatterns*), 270
SuiteRunner (*class in robot.running.suiterunner*), 585
suites (*robot.model.suitestatistics.SuiteStatistics attribute*), 276
suites (*robot.model.testsuite.TestSuite attribute*), 285
suites (*robot.result.model.TestSuite attribute*), 498
suites (*robot.running.model.TestSuite attribute*), 575
SuiteSetup (*class in robot.parsing.model.statements*), 357
SuiteStat (*class in robot.model.stats*), 275
SuiteStatistics (*class in robot.model.suitestatistics*), 276
SuiteStatisticsBuilder (*class in robot.model.suitestatistics*), 276
SuiteStatus (*class in robot.running.status*), 583

SuiteStructure (*class in robot.parsing.suitestructure*), 392
SuiteStructureBuilder (*class in robot.parsing.suitestructure*), 392
SuiteStructureParser (*class in robot.running.builder.builders*), 542
SuiteStructureVisitor (*class in robot.parsing.suitestructure*), 393
SuiteTeardown (*class in robot.parsing.model.statements*), 358
SuiteTeardownFailed (*class in robot.result.suiteteardownfailed*), 509
SuiteTeardownFailureHandler (*class in robot.result.suiteteardownfailed*), 505
SuiteVisitor (*class in robot.model.visitor*), 294
SuiteWriter (*class in robot.reporting.jswriter*), 395
supports_embedded_args (*robot.running.handlers.EmbeddedArgumentsHandler attribute*), 551
supports_embedded_args (*robot.running.userkeyword.EmbeddedArgumentsHandler attribute*), 590
supports_embedded_args (*robot.running.userkeyword.UserKeywordHandler attribute*), 590
supports_embedded_arguments (*robot.running.usererrorhandler.UserErrorHandler attribute*), 589
supports_kwargs (*robot.running.dynamicmethods.RunKeyword attribute*), 551
Sv (*class in robot.conf.languages*), 57
switch() (*robot.utils.connectioncache.ConnectionCache method*), 596
switch_connection() (*robot.libraries.Telnet.Telnet method*), 141
switch_process() (*robot.libraries.Process.Process method*), 127
system_decode() (*in module robot.utils.encoding*), 597
system_encode() (*in module robot.utils.encoding*), 597

T

TableFormatter (*class in robot.utils.htmlformatters*), 600
tag (*robot.result.xmlelementhandlers.ArgumentHandler attribute*), 525
tag (*robot.result.xmlelementhandlers.ArgumentsHandler attribute*), 525
tag (*robot.result.xmlelementhandlers.AssignHandler attribute*), 525
tag (*robot.result.xmlelementhandlers.BranchHandler attribute*), 521
tag (*robot.result.xmlelementhandlers.BreakHandler attribute*), 522
tag (*robot.result.xmlelementhandlers.ContinueHandler attribute*), 522
tag (*robot.result.xmlelementhandlers.DocHandler attribute*), 523
tag (*robot.result.xmlelementhandlers.ElementHandler attribute*), 519
tag (*robot.result.xmlelementhandlers.ErrorMessageHandler attribute*), 526
tag (*robot.result.xmlelementhandlers.ErrorsHandler attribute*), 526
tag (*robot.result.xmlelementhandlers.ForHandler attribute*), 520
tag (*robot.result.xmlelementhandlers.IfHandler attribute*), 521
tag (*robot.result.xmlelementhandlers.IterationHandler attribute*), 521
tag (*robot.result.xmlelementhandlers.KeywordHandler attribute*), 520
tag (*robot.result.xmlelementhandlers.MessageHandler attribute*), 523
tag (*robot.result.xmlelementhandlers.MetadataHandler attribute*), 523
tag (*robot.result.xmlelementhandlers.MetadataItemHandler attribute*), 524
tag (*robot.result.xmlelementhandlers.MetaHandler attribute*), 524
tag (*robot.result.xmlelementhandlers.PatternHandler attribute*), 522
tag (*robot.result.xmlelementhandlers.ReturnHandler attribute*), 522
tag (*robot.result.xmlelementhandlers.RobotHandler attribute*), 519
tag (*robot.result.xmlelementhandlers.RootHandler attribute*), 519
tag (*robot.result.xmlelementhandlers.StatisticsHandler attribute*), 526
tag (*robot.result.xmlelementhandlers.StatusHandler attribute*), 523
tag (*robot.result.xmlelementhandlers.SuiteHandler attribute*), 519
tag (*robot.result.xmlelementhandlers.TagHandler attribute*), 524
tag (*robot.result.xmlelementhandlers.TagsHandler attribute*), 524
tag (*robot.result.xmlelementhandlers.TestHandler attribute*), 520
tag (*robot.result.xmlelementhandlers.TimeoutHandler attribute*), 524
tag (*robot.result.xmlelementhandlers.TryHandler attribute*), 521
tag (*robot.result.xmlelementhandlers.ValueHandler attribute*), 526

tag (*robot.result.xmlelementhandlers.VarHandler attribute*), 525
tag (*robot.result.xmlelementhandlers.WhileHandler attribute*), 520
TagHandler (*class in robot.result.xmlelementhandlers*), 524
TagPattern() (*in module robot.model.tags*), 277
TagPatterns (*class in robot.model.tags*), 277
Tags (*class in robot.model.tags*), 277
Tags (*class in robot.parsing.model.statements*), 366
tags (*robot.model.keyword.Keyword attribute*), 260
tags (*robot.model.statistics.Statistics attribute*), 270
tags (*robot.model.tagstatistics.TagStatistics attribute*), 282
tags (*robot.model.testcase.TestCase attribute*), 283
TAGS (*robot.parsing.lexer.tokens.END attribute*), 339
TAGS (*robot.parsing.lexer.tokens.EOS attribute*), 337
TAGS (*robot.parsing.lexer.tokens.Token attribute*), 334
tags (*robot.result.model.Break attribute*), 490
tags (*robot.result.model.Continue attribute*), 487
tags (*robot.result.model.For attribute*), 469
tags (*robot.result.model.ForIteration attribute*), 467
tags (*robot.result.model.If attribute*), 478
tags (*robot.result.model.IfBranch attribute*), 476
tags (*robot.result.model.Keyword attribute*), 492
tags (*robot.result.model.Return attribute*), 485
tags (*robot.result.model.TestCase attribute*), 495
tags (*robot.result.model.Try attribute*), 483
tags (*robot.result.model.TryBranch attribute*), 480
tags (*robot.result.model.While attribute*), 474
tags (*robot.result.model.WhileIteration attribute*), 471
tags (*robot.result.modeldeprecation.DeprecatedAttributesMixin attribute*), 500
tags (*robot.running.builder.settings.TestSettings attribute*), 544
tags (*robot.running.model.Keyword attribute*), 557
tags (*robot.running.model.TestCase attribute*), 571
tags (*robot.running.model.UserKeyword attribute*), 576
tags (*robot.running.userkeywordrunner.EmbeddedArgumentsRunner attribute*), 590
tags (*robot.running.userkeywordrunner.UserKeywordRunner attribute*), 590
tags_setting (*robot.conf.languages.Bg attribute*), 60
tags_setting (*robot.conf.languages.Bs attribute*), 39
tags_setting (*robot.conf.languages.Cs attribute*), 36
tags_setting (*robot.conf.languages.De attribute*), 43
tags_setting (*robot.conf.languages.En attribute*), 35
tags_setting (*robot.conf.languages.Es attribute*), 51
tags_setting (*robot.conf.languages.Fi attribute*), 40
tags_setting (*robot.conf.languages.Fr attribute*), 41
tags_setting (*robot.conf.languages.Hi attribute*), 64
tags_setting (*robot.conf.languages.It attribute*), 62
tags_setting (*robot.conf.languages.Language attribute*), 33
tags_setting (*robot.conf.languages.Nl attribute*), 37
tags_setting (*robot.conf.languages.Pl attribute*), 48
tags_setting (*robot.conf.languages.Pt attribute*), 46
tags_setting (*robot.conf.languages.PtBr attribute*), 44
tags_setting (*robot.conf.languages.Ro attribute*), 61
tags_setting (*robot.conf.languages.Ru attribute*), 53
tags_setting (*robot.conf.languages.Sv attribute*), 58
tags_setting (*robot.conf.languages.Th attribute*), 47
tags_setting (*robot.conf.languages.Tr attribute*), 57
tags_setting (*robot.conf.languages.Uk attribute*), 50
tags_setting (*robot.conf.languages.ZhCn attribute*), 54
tags_setting (*robot.conf.languages.ZhTw attribute*), 55
TagSetter (*class in robot.model.tagsetter*), 277
TagsHandler (*class in robot.result.xmlelementhandlers*), 524
TagStat (*class in robot.model.stats*), 275
TagStatDoc (*class in robot.model.tagstatistics*), 282
TagStatInfo (*class in robot.model.tagstatistics*), 282
TagStatistics (*class in robot.model.tagstatistics*), 282
TagStatisticsBuilder (*class in robot.model.tagstatistics*), 282
TagStatLink (*class in robot.model.tagstatistics*), 282
take_screenshot() (*robot.libraries.Screenshot.Screenshot method*), 131
take_screenshot_without_embedding() (*robot.libraries.Screenshot.Screenshot method*), 131
TASK_HEADER (*robot.parsing.lexer.tokens.END attribute*), 339
TASK_HEADER (*robot.parsing.lexer.tokens.EOS attribute*), 337
TASK_HEADER (*robot.parsing.lexer.tokens.Token attribute*), 334
task_section() (*robot.parsing.lexer.context.FileContext method*), 323
task_section() (*robot.parsing.lexer.context.InitFileContext method*), 325
task_section() (*robot.parsing.lexer.context.ResourceFileContext method*), 324
task_section() (*robot.parsing.lexer.context.TestCaseFileContext method*), 324
task_setup_setting (*robot.conf.languages.Bg attribute*), 59
task_setup_setting (*robot.conf.languages.Bs attribute*), 39
task_setup_setting (*robot.conf.languages.Cs attribute*), 36
task_setup_setting (*robot.conf.languages.De attribute*), 43

task_setup_setting (*robot.conf.languages.En attribute*), 34
task_setup_setting (*robot.conf.languages.Es attribute*), 51
task_setup_setting (*robot.conf.languages.Fi attribute*), 40
task_setup_setting (*robot.conf.languages.Fr attribute*), 41
task_setup_setting (*robot.conf.languages.Hi attribute*), 64
task_setup_setting (*robot.conf.languages.It attribute*), 62
task_setup_setting (*robot.conf.languages.Language attribute*), 33
task_setup_setting (*robot.conf.languages.Nl attribute*), 37
task_setup_setting (*robot.conf.languages.Pl attribute*), 48
task_setup_setting (*robot.conf.languages.Pt attribute*), 45
task_setup_setting (*robot.conf.languages.PtBr attribute*), 44
task_setup_setting (*robot.conf.languages.Ro attribute*), 61
task_setup_setting (*robot.conf.languages.Ru attribute*), 52
task_setup_setting (*robot.conf.languages.Sv attribute*), 58
task_setup_setting (*robot.conf.languages.Th attribute*), 47
task_setup_setting (*robot.conf.languages.Tr attribute*), 57
task_setup_setting (*robot.conf.languages.Uk attribute*), 50
task_setup_setting (*robot.conf.languages.ZhCn attribute*), 54
task_setup_setting (*robot.conf.languages.ZhTw attribute*), 55
task_tags_setting (*robot.conf.languages.Bg attribute*), 60
task_tags_setting (*robot.conf.languages.Bs attribute*), 39
task_tags_setting (*robot.conf.languages.Cs attribute*), 36
task_tags_setting (*robot.conf.languages.De attribute*), 43
task_tags_setting (*robot.conf.languages.En attribute*), 34
task_tags_setting (*robot.conf.languages.Es attribute*), 51
task_tags_setting (*robot.conf.languages.Fi attribute*), 40
task_tags_setting (*robot.conf.languages.Fr attribute*), 41
task_tags_setting (*robot.conf.languages.Hi attribute*), 64
task_tags_setting (*robot.conf.languages.It attribute*), 62
task_tags_setting (*robot.conf.languages.Language attribute*), 33
task_tags_setting (*robot.conf.languages.Nl attribute*), 37
task_tags_setting (*robot.conf.languages.Pl attribute*), 48
task_tags_setting (*robot.conf.languages.Pt attribute*), 46
task_tags_setting (*robot.conf.languages.PtBr attribute*), 44
task_tags_setting (*robot.conf.languages.Ro attribute*), 61
task_tags_setting (*robot.conf.languages.Ru attribute*), 53
task_tags_setting (*robot.conf.languages.Sv attribute*), 58
task_tags_setting (*robot.conf.languages.Th attribute*), 47
task_tags_setting (*robot.conf.languages.Tr attribute*), 57
task_tags_setting (*robot.conf.languages.Uk attribute*), 50
task_tags_setting (*robot.conf.languages.ZhCn attribute*), 54
task_tags_setting (*robot.conf.languages.ZhTw attribute*), 55
task_teardown_setting (*robot.conf.languages.Bg attribute*), 59
task_teardown_setting (*robot.conf.languages.Bs attribute*), 39
task_teardown_setting (*robot.conf.languages.Cs attribute*), 36
task_teardown_setting (*robot.conf.languages.De attribute*), 43
task_teardown_setting (*robot.conf.languages.En attribute*), 34
task_teardown_setting (*robot.conf.languages.Es attribute*), 51
task_teardown_setting (*robot.conf.languages.Fi attribute*), 40
task_teardown_setting (*robot.conf.languages.Fr attribute*), 41
task_teardown_setting (*robot.conf.languages.Hi attribute*), 64
task_teardown_setting (*robot.conf.languages.It attribute*), 62
task_teardown_setting (*robot.conf.languages.Language attribute*),

33
task_teardown_setting (*robot.conf.languages.Nl attribute*), 37
task_teardown_setting (*robot.conf.languages.Pl attribute*), 48
task_teardown_setting (*robot.conf.languages.Pt attribute*), 46
task_teardown_setting (*robot.conf.languages.PtBr attribute*), 44
task_teardown_setting (*robot.conf.languages.Ro attribute*), 61
task_teardown_setting (*robot.conf.languages.Ru attribute*), 53
task_teardown_setting (*robot.conf.languages.Sv attribute*), 58
task_teardown_setting (*robot.conf.languages.Th attribute*), 47
task_teardown_setting (*robot.conf.languages.Tr attribute*), 57
task_teardown_setting (*robot.conf.languages.Uk attribute*), 50
task_teardown_setting (*robot.conf.languages.ZhCn attribute*), 54
task_teardown_setting (*robot.conf.languages.ZhTw attribute*), 55
task_template_setting (*robot.conf.languages.Bg attribute*), 60
task_template_setting (*robot.conf.languages.Bs attribute*), 39
task_template_setting (*robot.conf.languages.Cs attribute*), 36
task_template_setting (*robot.conf.languages.De attribute*), 43
task_template_setting (*robot.conf.languages.En attribute*), 34
task_template_setting (*robot.conf.languages.Es attribute*), 51
task_template_setting (*robot.conf.languages.Fi attribute*), 40
task_template_setting (*robot.conf.languages.Fr attribute*), 41
task_template_setting (*robot.conf.languages.Hi attribute*), 64
task_template_setting (*robot.conf.languages.It attribute*), 62
task_template_setting (*robot.conf.languages.Language attribute*), 33
task_template_setting (*robot.conf.languages.Nl attribute*), 37
task_template_setting (*robot.conf.languages.Pl attribute*), 48
task_template_setting (*robot.conf.languages.Pt attribute*), 46
task_template_setting (*robot.conf.languages.PtBr attribute*), 44
task_template_setting (*robot.conf.languages.Ro attribute*), 61
task_template_setting (*robot.conf.languages.Ru attribute*), 53
task_template_setting (*robot.conf.languages.Sv attribute*), 58
task_template_setting (*robot.conf.languages.Th attribute*), 47
task_template_setting (*robot.conf.languages.Tr attribute*), 57
task_template_setting (*robot.conf.languages.Uk attribute*), 50
task_template_setting (*robot.conf.languages.ZhCn attribute*), 54
task_template_setting (*robot.conf.languages.ZhTw attribute*), 55
task_timeout_setting (*robot.conf.languages.Bg attribute*), 60
task_timeout_setting (*robot.conf.languages.Bs attribute*), 39
task_timeout_setting (*robot.conf.languages.Cs attribute*), 36
task_timeout_setting (*robot.conf.languages.De attribute*), 43
task_timeout_setting (*robot.conf.languages.En attribute*), 34
task_timeout_setting (*robot.conf.languages.Es attribute*), 51
task_timeout_setting (*robot.conf.languages.Fi attribute*), 40
task_timeout_setting (*robot.conf.languages.Fr attribute*), 41
task_timeout_setting (*robot.conf.languages.Hi attribute*), 64
task_timeout_setting (*robot.conf.languages.It attribute*), 62
task_timeout_setting (*robot.conf.languages.Language attribute*), 33
task_timeout_setting (*robot.conf.languages.Nl attribute*), 37
task_timeout_setting (*robot.conf.languages.Pl attribute*), 48
task_timeout_setting (*robot.conf.languages.Pt attribute*), 46
task_timeout_setting (*robot.conf.languages.PtBr attribute*), 44
task_timeout_setting (*robot.conf.languages.Ro attribute*), 61
task_timeout_setting (*robot.conf.languages.Ru attribute*), 53
task_timeout_setting (*robot.conf.languages.Sv*

attribute), 58
task_timeout_setting (*robot.conf.languages.Th attribute*), 47
task_timeout_setting (*robot.conf.languages.Tr attribute*), 57
task_timeout_setting (*robot.conf.languages.Uk attribute*), 50
task_timeout_setting (*robot.conf.languages.ZhCn attribute*), 54
task_timeout_setting (*robot.conf.languages.ZhTw attribute*), 55
tasks (*robot.parsing.model.blocks.TestCaseSection attribute*), 342
tasks_header (*robot.conf.languages.Bg attribute*), 59
tasks_header (*robot.conf.languages.Bs attribute*), 38
tasks_header (*robot.conf.languages.Cs attribute*), 35
tasks_header (*robot.conf.languages.De attribute*), 42
tasks_header (*robot.conf.languages.En attribute*), 34
tasks_header (*robot.conf.languages.Es attribute*), 51
tasks_header (*robot.conf.languages.Fi attribute*), 40
tasks_header (*robot.conf.languages.Fr attribute*), 41
tasks_header (*robot.conf.languages.Hi attribute*), 63
tasks_header (*robot.conf.languages.It attribute*), 62
tasks_header (*robot.conf.languages.Language attribute*), 32
tasks_header (*robot.conf.languages.Nl attribute*), 37
tasks_header (*robot.conf.languages.Pl attribute*), 48
tasks_header (*robot.conf.languages.Pt attribute*), 45
tasks_header (*robot.conf.languages.PtBr attribute*), 44
tasks_header (*robot.conf.languages.Ro attribute*), 60
tasks_header (*robot.conf.languages.Ru attribute*), 52
tasks_header (*robot.conf.languages.Sv attribute*), 58
tasks_header (*robot.conf.languages.Th attribute*), 46
tasks_header (*robot.conf.languages.Tr attribute*), 56
tasks_header (*robot.conf.languages.Uk attribute*), 49
tasks_header (*robot.conf.languages.ZhCn attribute*), 53
tasks_header (*robot.conf.languages.ZhTw attribute*), 55
TaskSectionHeaderLexer (*class in robot.parsing.lexer.statementlexers*), 329
TaskSectionLexer (*class in robot.parsing.lexer.blocklexers*), 320
Teardown (*class in robot.parsing.model.statements*), 365
TEARDOWN (*robot.model.body.BodyItem attribute*), 228
TEARDOWN (*robot.model.control.Break attribute*), 249
TEARDOWN (*robot.model.control.Continue attribute*), 248
TEARDOWN (*robot.model.control.For attribute*), 239
TEARDOWN (*robot.model.control.If attribute*), 243
TEARDOWN (*robot.model.control.IfBranch attribute*), 242
TEARDOWN (*robot.model.control.Return attribute*), 247
TEARDOWN (*robot.model.control.Try attribute*), 246
TEARDOWN (*robot.model.control.TryBranch attribute*), 244
TEARDOWN (*robot.model.control.While attribute*), 240
TEARDOWN (*robot.model.keyword.Keyword attribute*), 261
teardown (*robot.model.keyword.Keyword attribute*), 259
teardown (*robot.model.keyword.Keywords attribute*), 261
TEARDOWN (*robot.model.message.Message attribute*), 263
teardown (*robot.model.testcase.TestCase attribute*), 283
teardown (*robot.model.testsuite.TestSuite attribute*), 286
TEARDOWN (*robot.output.loggerhelper.Message attribute*), 310
TEARDOWN (*robot.parsing.lexer.tokens.END attribute*), 339
TEARDOWN (*robot.parsing.lexer.tokens.EOS attribute*), 337
TEARDOWN (*robot.parsing.lexer.tokens.Token attribute*), 334
TEARDOWN (*robot.result.model.Break attribute*), 488
TEARDOWN (*robot.result.model.Continue attribute*), 486
TEARDOWN (*robot.result.model.For attribute*), 468
TEARDOWN (*robot.result.model.ForIteration attribute*), 465
TEARDOWN (*robot.result.model.If attribute*), 477
TEARDOWN (*robot.result.model.IfBranch attribute*), 475
TEARDOWN (*robot.result.model.Keyword attribute*), 491
teardown (*robot.result.model.Keyword attribute*), 492
TEARDOWN (*robot.result.model.Message attribute*), 463
TEARDOWN (*robot.result.model.Return attribute*), 484
teardown (*robot.result.model.TestCase attribute*), 495
teardown (*robot.result.model.TestSuite attribute*), 498
TEARDOWN (*robot.result.model.Try attribute*), 481
TEARDOWN (*robot.result.model.TryBranch attribute*), 479
TEARDOWN (*robot.result.model.While attribute*), 472
TEARDOWN (*robot.result.model.WhileIteration attribute*), 470
teardown (*robot.running.builder.settings.Defaults attribute*), 543
teardown (*robot.running.builder.settings.TestSettings attribute*), 543
TEARDOWN (*robot.running.model.Break attribute*), 569
TEARDOWN (*robot.running.model.Continue attribute*), 568
TEARDOWN (*robot.running.model.For attribute*), 558
TEARDOWN (*robot.running.model.If attribute*), 562
TEARDOWN (*robot.running.model.IfBranch attribute*), 561

TEARDOWN (*robot.running.model.Keyword attribute*), 556
teardown (*robot.running.model.Keyword attribute*), 557
TEARDOWN (*robot.running.model.Return attribute*), 566
teardown (*robot.running.model.TestCase attribute*), 572
teardown (*robot.running.model.TestSuite attribute*), 575
TEARDOWN (*robot.running.model.Try attribute*), 565
TEARDOWN (*robot.running.model.TryBranch attribute*), 563
teardown (*robot.running.model.UserKeyword attribute*), 576
TEARDOWN (*robot.running.model.While attribute*), 559
teardown_allowed (*robot.running.status.Exit attribute*), 583
teardown_allowed (*robot.running.status.SuiteStatus attribute*), 583
teardown_allowed (*robot.running.status.TestStatus attribute*), 583
teardown_executed() (*robot.running.status.SuiteStatus method*), 583
teardown_executed() (*robot.running.status.TestStatus method*), 583
teardown_message (*robot.running.status.ParentMessage attribute*), 584
teardown_message (*robot.running.status.SuiteMessage attribute*), 584
teardown_message (*robot.running.status.TestMessage attribute*), 584
teardown_setting (*robot.conf.languages.Bg attribute*), 60
teardown_setting (*robot.conf.languages.Bs attribute*), 39
teardown_setting (*robot.conf.languages.Cs attribute*), 36
teardown_setting (*robot.conf.languages.De attribute*), 43
teardown_setting (*robot.conf.languages.En attribute*), 35
teardown_setting (*robot.conf.languages.Es attribute*), 51
teardown_setting (*robot.conf.languages.Fi attribute*), 40
teardown_setting (*robot.conf.languages.Fr attribute*), 42
teardown_setting (*robot.conf.languages.Hi attribute*), 64
teardown_setting (*robot.conf.languages.It attribute*), 63
teardown_setting (*robot.conf.languages.Language attribute*), 33
teardown_setting (*robot.conf.languages.Nl attribute*), 37
teardown_setting (*robot.conf.languages.Pl attribute*), 49
teardown_setting (*robot.conf.languages.Pt attribute*), 46
teardown_setting (*robot.conf.languages.PtBr attribute*), 44
teardown_setting (*robot.conf.languages.Ro attribute*), 61
teardown_setting (*robot.conf.languages.Ru attribute*), 53
teardown_setting (*robot.conf.languages.Sv attribute*), 58
teardown_setting (*robot.conf.languages.Th attribute*), 47
teardown_setting (*robot.conf.languages.Tr attribute*), 57
teardown_setting (*robot.conf.languages.Uk attribute*), 50
teardown_setting (*robot.conf.languages.ZhCn attribute*), 54
teardown_setting (*robot.conf.languages.ZhTw attribute*), 56
teardown_skipped_message (*robot.running.status.ParentMessage attribute*), 584
teardown_skipped_message (*robot.running.status.SuiteMessage attribute*), 584
teardown_skipped_message (*robot.running.status.TestMessage attribute*), 584
Telnet (*class in robot.libraries.Telnet*), 137
TelnetConnection (*class in robot.libraries.Telnet*), 141
Template (*class in robot.parsing.model.statements*), 367
TEMPLATE (*robot.parsing.lexer.tokens.END attribute*), 339
TEMPLATE (*robot.parsing.lexer.tokens.EOS attribute*), 337
TEMPLATE (*robot.parsing.lexer.tokens.Token attribute*), 334
template (*robot.running.builder.settings.TestSettings attribute*), 543
template (*robot.running.model.TestCase attribute*), 570
template_set (*robot.parsing.lexer.context.KeywordContext attribute*), 325
template_set (*robot.parsing.lexer.context.TestCaseContext attribute*), 325
template_set (*robot.parsing.lexer.settings.TestCaseSettings*

attribute), 327
template_setting (*robot.conf.languages.Bg attribute*), 60
template_setting (*robot.conf.languages.Bs attribute*), 39
template_setting (*robot.conf.languages.Cs attribute*), 36
template_setting (*robot.conf.languages.De attribute*), 43
template_setting (*robot.conf.languages.En attribute*), 35
template_setting (*robot.conf.languages.Es attribute*), 51
template_setting (*robot.conf.languages.Fi attribute*), 40
template_setting (*robot.conf.languages.Fr attribute*), 42
template_setting (*robot.conf.languages.Hi attribute*), 64
template_setting (*robot.conf.languages.It attribute*), 63
template_setting (*robot.conf.languages.Language attribute*), 33
template_setting (*robot.conf.languages.Nl attribute*), 37
template_setting (*robot.conf.languages.Pl attribute*), 49
template_setting (*robot.conf.languages.Pt attribute*), 46
template_setting (*robot.conf.languages.PtBr attribute*), 44
template_setting (*robot.conf.languages.Ro attribute*), 61
template_setting (*robot.conf.languages.Ru attribute*), 53
template_setting (*robot.conf.languages.Sv attribute*), 58
template_setting (*robot.conf.languages.Th attribute*), 47
template_setting (*robot.conf.languages.Tr attribute*), 57
template_setting (*robot.conf.languages.Uk attribute*), 50
template_setting (*robot.conf.languages.ZhCn attribute*), 54
template_setting (*robot.conf.languages.ZhTw attribute*), 56
TemplateArguments (*class in robot.parsing.model.statements*), 371
TerminalEmulator (*class in robot.libraries.Telnet*), 146
terminate_all_processes() (*robot.libraries.Process.Process method*), 126
terminate_process() (*robot.libraries.Process.Process method*), 126
TERMINATE_TIMEOUT (*robot.libraries.Process.Process attribute*), 125
test() (*robot.libraries.Screenshot.ScreenshotTaker method*), 131
test_case_context() (*robot.parsing.lexer.context.TestCaseFileContext method*), 324
test_case_section() (*robot.parsing.lexer.context.FileContext method*), 323
test_case_section() (*robot.parsing.lexer.context.InitFileContext method*), 325
test_case_section() (*robot.parsing.lexer.context.ResourceFileContext method*), 324
test_case_section() (*robot.parsing.lexer.context.TestCaseFileContext method*), 324
test_cases_header (*robot.conf.languages.Bg attribute*), 59
test_cases_header (*robot.conf.languages.Bs attribute*), 38
test_cases_header (*robot.conf.languages.Cs attribute*), 35
test_cases_header (*robot.conf.languages.De attribute*), 42
test_cases_header (*robot.conf.languages.En attribute*), 34
test_cases_header (*robot.conf.languages.Es attribute*), 51
test_cases_header (*robot.conf.languages.Fi attribute*), 39
test_cases_header (*robot.conf.languages.Fr attribute*), 41
test_cases_header (*robot.conf.languages.Hi attribute*), 63
test_cases_header (*robot.conf.languages.It attribute*), 62
test_cases_header (*robot.conf.languages.Language attribute*), 32
test_cases_header (*robot.conf.languages.Nl attribute*), 37
test_cases_header (*robot.conf.languages.Pl attribute*), 48
test_cases_header (*robot.conf.languages.Pt attribute*), 45
test_cases_header (*robot.conf.languages.PtBr attribute*), 44

test_cases_header (*robot.conf.languages.Ro attribute*), 60
test_cases_header (*robot.conf.languages.Ru attribute*), 52
test_cases_header (*robot.conf.languages.Sv attribute*), 58
test_cases_header (*robot.conf.languages.Th attribute*), 46
test_cases_header (*robot.conf.languages.Tr attribute*), 56
test_cases_header (*robot.conf.languages.Uk attribute*), 49
test_cases_header (*robot.conf.languages.ZhCn attribute*), 53
test_cases_header (*robot.conf.languages.ZhTw attribute*), 55
test_class (*robot.model.testsuite.TestSuite attribute*), 285
test_class (*robot.result.model.TestSuite attribute*), 496
test_class (*robot.running.model.TestSuite attribute*), 572
test_count (*robot.model.testsuite.TestSuite attribute*), 286
test_count (*robot.result.model.TestSuite attribute*), 498
test_count (*robot.running.model.TestSuite attribute*), 576
test_failed() (*robot.running.status.TestStatus method*), 583
test_names (*robot.conf.settings.RebotSettings attribute*), 67
test_names (*robot.conf.settings.RobotSettings attribute*), 65
test_or_task() (*in module robot.utils.misc*), 604
test_separator() (*robot.output.console.verbose.VerboseWriter method*), 304
TEST_SETUP (*robot.parsing.lexer.tokens.END attribute*), 339
TEST_SETUP (*robot.parsing.lexer.tokens.EOS attribute*), 337
TEST_SETUP (*robot.parsing.lexer.tokens.Token attribute*), 334
test_setup_setting (*robot.conf.languages.Bg attribute*), 59
test_setup_setting (*robot.conf.languages.Bs attribute*), 38
test_setup_setting (*robot.conf.languages.Cs attribute*), 36
test_setup_setting (*robot.conf.languages.De attribute*), 43
test_setup_setting (*robot.conf.languages.En attribute*), 34
test_setup_setting (*robot.conf.languages.Es attribute*), 51
test_setup_setting (*robot.conf.languages.Fi attribute*), 40
test_setup_setting (*robot.conf.languages.Fr attribute*), 41
test_setup_setting (*robot.conf.languages.Hi attribute*), 64
test_setup_setting (*robot.conf.languages.It attribute*), 62
test_setup_setting (*robot.conf.languages.Language attribute*), 33
test_setup_setting (*robot.conf.languages.Nl attribute*), 37
test_setup_setting (*robot.conf.languages.Pl attribute*), 48
test_setup_setting (*robot.conf.languages.Pt attribute*), 45
test_setup_setting (*robot.conf.languages.PtBr attribute*), 44
test_setup_setting (*robot.conf.languages.Ro attribute*), 61
test_setup_setting (*robot.conf.languages.Ru attribute*), 52
test_setup_setting (*robot.conf.languages.Sv attribute*), 58
test_setup_setting (*robot.conf.languages.Th attribute*), 47
test_setup_setting (*robot.conf.languages.Tr attribute*), 57
test_setup_setting (*robot.conf.languages.Uk attribute*), 50
test_setup_setting (*robot.conf.languages.ZhCn attribute*), 54
test_setup_setting (*robot.conf.languages.ZhTw attribute*), 55
test_skipped() (*robot.running.status.TestStatus method*), 583
test_tags_setting (*robot.conf.languages.Bg attribute*), 59
test_tags_setting (*robot.conf.languages.Bs attribute*), 39
test_tags_setting (*robot.conf.languages.Cs attribute*), 36
test_tags_setting (*robot.conf.languages.De attribute*), 43
test_tags_setting (*robot.conf.languages.En attribute*), 34
test_tags_setting (*robot.conf.languages.Es attribute*), 51
test_tags_setting (*robot.conf.languages.Fi attribute*), 40
test_tags_setting (*robot.conf.languages.Fr attribute*), 41

test_tags_setting (*robot.conf.languages.Hi attribute*), 64
test_tags_setting (*robot.conf.languages.It attribute*), 62
test_tags_setting (*robot.conf.languages.Language attribute*), 33
test_tags_setting (*robot.conf.languages.Nl attribute*), 37
test_tags_setting (*robot.conf.languages.Pl attribute*), 48
test_tags_setting (*robot.conf.languages.Pt attribute*), 45
test_tags_setting (*robot.conf.languages.PtBr attribute*), 44
test_tags_setting (*robot.conf.languages.Ro attribute*), 61
test_tags_setting (*robot.conf.languages.Ru attribute*), 52
test_tags_setting (*robot.conf.languages.Sv attribute*), 58
test_tags_setting (*robot.conf.languages.Th attribute*), 47
test_tags_setting (*robot.conf.languages.Tr attribute*), 57
test_tags_setting (*robot.conf.languages.Uk attribute*), 50
test_tags_setting (*robot.conf.languages.ZhCn attribute*), 54
test_tags_setting (*robot.conf.languages.ZhTw attribute*), 55
TEST_TEARDOWN (*robot.parsing.lexer.tokens.END attribute*), 339
TEST_TEARDOWN (*robot.parsing.lexer.tokens.EOS attribute*), 337
TEST_TEARDOWN (*robot.parsing.lexer.tokens.Token attribute*), 334
test_teardown_setting (*robot.conf.languages.Bg attribute*), 59
test_teardown_setting (*robot.conf.languages.Bs attribute*), 38
test_teardown_setting (*robot.conf.languages.Cs attribute*), 36
test_teardown_setting (*robot.conf.languages.De attribute*), 43
test_teardown_setting (*robot.conf.languages.En attribute*), 34
test_teardown_setting (*robot.conf.languages.Es attribute*), 51
test_teardown_setting (*robot.conf.languages.Fi attribute*), 40
test_teardown_setting (*robot.conf.languages.Fr attribute*), 41
test_teardown_setting (*robot.conf.languages.Hi attribute*), 64
test_teardown_setting (*robot.conf.languages.It attribute*), 62
test_teardown_setting (*robot.conf.languages.Language attribute*), 33
test_teardown_setting (*robot.conf.languages.Nl attribute*), 37
test_teardown_setting (*robot.conf.languages.Pl attribute*), 48
test_teardown_setting (*robot.conf.languages.Pt attribute*), 45
test_teardown_setting (*robot.conf.languages.PtBr attribute*), 44
test_teardown_setting (*robot.conf.languages.Ro attribute*), 61
test_teardown_setting (*robot.conf.languages.Ru attribute*), 52
test_teardown_setting (*robot.conf.languages.Sv attribute*), 58
test_teardown_setting (*robot.conf.languages.Th attribute*), 47
test_teardown_setting (*robot.conf.languages.Tr attribute*), 57
test_teardown_setting (*robot.conf.languages.Uk attribute*), 50
test_teardown_setting (*robot.conf.languages.ZhCn attribute*), 54
test_teardown_setting (*robot.conf.languages.ZhTw attribute*), 55
TEST_TEMPLATE (*robot.parsing.lexer.tokens.END attribute*), 339
TEST_TEMPLATE (*robot.parsing.lexer.tokens.EOS attribute*), 337
TEST_TEMPLATE (*robot.parsing.lexer.tokens.Token attribute*), 334
test_template_setting (*robot.conf.languages.Bg attribute*), 59
test_template_setting (*robot.conf.languages.Bs attribute*), 38
test_template_setting (*robot.conf.languages.Cs attribute*), 36
test_template_setting (*robot.conf.languages.De attribute*), 43
test_template_setting (*robot.conf.languages.En attribute*), 34
test_template_setting (*robot.conf.languages.Es attribute*), 51
test_template_setting (*robot.conf.languages.Fi attribute*), 40
test_template_setting (*robot.conf.languages.Fr attribute*), 41
test_template_setting (*robot.conf.languages.Hi attribute*), 64

test_template_setting (*robot.conf.languages.It attribute*), 62
test_template_setting (*robot.conf.languages.Language attribute*), 33
test_template_setting (*robot.conf.languages.Nl attribute*), 37
test_template_setting (*robot.conf.languages.Pl attribute*), 48
test_template_setting (*robot.conf.languages.Pt attribute*), 45
test_template_setting (*robot.conf.languages.PtBr attribute*), 44
test_template_setting (*robot.conf.languages.Ro attribute*), 61
test_template_setting (*robot.conf.languages.Ru attribute*), 52
test_template_setting (*robot.conf.languages.Sv attribute*), 58
test_template_setting (*robot.conf.languages.Th attribute*), 47
test_template_setting (*robot.conf.languages.Tr attribute*), 57
test_template_setting (*robot.conf.languages.Uk attribute*), 50
test_template_setting (*robot.conf.languages.ZhCn attribute*), 54
test_template_setting (*robot.conf.languages.ZhTw attribute*), 55
TEST_TIMEOUT (*robot.parsing.lexer.tokens.END attribute*), 339
TEST_TIMEOUT (*robot.parsing.lexer.tokens.EOS attribute*), 337
TEST_TIMEOUT (*robot.parsing.lexer.tokens.Token attribute*), 334
test_timeout_setting (*robot.conf.languages.Bg attribute*), 59
test_timeout_setting (*robot.conf.languages.Bs attribute*), 38
test_timeout_setting (*robot.conf.languages.Cs attribute*), 36
test_timeout_setting (*robot.conf.languages.De attribute*), 43
test_timeout_setting (*robot.conf.languages.En attribute*), 34
test_timeout_setting (*robot.conf.languages.Es attribute*), 51
test_timeout_setting (*robot.conf.languages.Fi attribute*), 40
test_timeout_setting (*robot.conf.languages.Fr attribute*), 41
test_timeout_setting (*robot.conf.languages.Hi attribute*), 64
test_timeout_setting (*robot.conf.languages.It attribute*), 62
test_timeout_setting (*robot.conf.languages.Language attribute*), 33
test_timeout_setting (*robot.conf.languages.Nl attribute*), 37
test_timeout_setting (*robot.conf.languages.Pl attribute*), 48
test_timeout_setting (*robot.conf.languages.Pt attribute*), 45
test_timeout_setting (*robot.conf.languages.PtBr attribute*), 44
test_timeout_setting (*robot.conf.languages.Ro attribute*), 61
test_timeout_setting (*robot.conf.languages.Ru attribute*), 52
test_timeout_setting (*robot.conf.languages.Sv attribute*), 58
test_timeout_setting (*robot.conf.languages.Th attribute*), 47
test_timeout_setting (*robot.conf.languages.Tr attribute*), 57
test_timeout_setting (*robot.conf.languages.Uk attribute*), 50
test_timeout_setting (*robot.conf.languages.ZhCn attribute*), 54
test_timeout_setting (*robot.conf.languages.ZhTw attribute*), 55
TestBuilder (*class in robot.reporting.jsmodelbuilders*), 394
TestCase (*class in robot.model.testcase*), 282
TestCase (*class in robot.parsing.model.blocks*), 342
TestCase (*class in robot.result.model*), 493
TestCase (*class in robot.running.model*), 570
TESTCASE_HEADER (*robot.parsing.lexer.tokens.END attribute*), 339
TESTCASE_HEADER (*robot.parsing.lexer.tokens.EOS attribute*), 337
TESTCASE_HEADER (*robot.parsing.lexer.tokens.Token attribute*), 334
TESTCASE_NAME (*robot.parsing.lexer.tokens.END attribute*), 339
TESTCASE_NAME (*robot.parsing.lexer.tokens.EOS attribute*), 337
TESTCASE_NAME (*robot.parsing.lexer.tokens.Token attribute*), 334
TestCaseBuilder (*class in robot.running.builder.transformers*), 545
TestCaseContext (*class in robot.parsing.lexer.context*), 325
TestCaseFileContext (*class in robot.parsing.lexer.context*), 324
TestCaseFileSettings (*class in robot.parsing.lexer.settings*), 326

TestCaseLexer (*class in robot.parsing.lexer.blocklexers*), 321
TestCaseName (*class in robot.parsing.model.statements*), 363
TestCaseParser (*class in robot.parsing.parser.blockparsers*), 389
TestCases (*class in robot.model.testcase*), 284
TestCaseScope (*class in robot.running.libraryscopes*), 553
TestCaseSection (*class in robot.parsing.model.blocks*), 342
TestCaseSectionHeaderLexer (*class in robot.parsing.lexer.statementlexers*), 329
TestCaseSectionLexer (*class in robot.parsing.lexer.blocklexers*), 319
TestCaseSectionParser (*class in robot.parsing.parser.fileparser*), 391
TestCaseSettings (*class in robot.parsing.lexer.settings*), 327
TestDoc (*class in robot.testdoc*), 624
testdoc() (*in module robot.testdoc*), 624
testdoc_cli() (*in module robot.testdoc*), 624
TestdocModelWriter (*class in robot.testdoc*), 624
TestHandler (*class in robot.result.xmlelementhandlers*), 520
TestLibrary() (*in module robot.running.testlibraries*), 589
TestMessage (*class in robot.running.status*), 584
TestNamePatterns (*class in robot.model.namepatterns*), 270
TestOrKeywordLexer (*class in robot.parsing.lexer.blocklexers*), 321
TestOrKeywordSettingLexer (*class in robot.parsing.lexer.statementlexers*), 330
tests (*robot.model.testsuite.TestSuite attribute*), 285
tests (*robot.result.model.TestSuite attribute*), 498
tests (*robot.running.model.TestSuite attribute*), 576
TestSettings (*class in robot.running.builder.settings*), 543
TestSetup (*class in robot.parsing.model.statements*), 359
TestStatus (*class in robot.running.status*), 583
TestSuite (*class in robot.model.testsuite*), 285
TestSuite (*class in robot.result.model*), 495
TestSuite (*class in robot.running.model*), 572
TestSuiteBuilder (*class in robot.running.builder.builders*), 541
TestSuiteFactory() (*in module robot.testdoc*), 624
TestSuites (*class in robot.model.testsuite*), 287
TestSuiteScope (*class in robot.running.libraryscopes*), 553
TestTeardown (*class in robot.parsing.model.statements*), 359
TestTemplate (*class in robot.parsing.model.statements*), 360
TestTimeout (*class in robot.parsing.model.statements*), 361
TestTimeout (*class in robot.running.timeouts*), 548
Th (*class in robot.conf.languages*), 46
then_prefixes (*robot.conf.languages.Bg attribute*), 60
then_prefixes (*robot.conf.languages.Bs attribute*), 39
then_prefixes (*robot.conf.languages.Cs attribute*), 36
then_prefixes (*robot.conf.languages.De attribute*), 43
then_prefixes (*robot.conf.languages.En attribute*), 35
then_prefixes (*robot.conf.languages.Es attribute*), 51
then_prefixes (*robot.conf.languages.Fi attribute*), 40
then_prefixes (*robot.conf.languages.Fr attribute*), 42
then_prefixes (*robot.conf.languages.Hi attribute*), 64
then_prefixes (*robot.conf.languages.It attribute*), 63
then_prefixes (*robot.conf.languages.Language attribute*), 33
then_prefixes (*robot.conf.languages.Nl attribute*), 38
then_prefixes (*robot.conf.languages.Pl attribute*), 49
then_prefixes (*robot.conf.languages.Pt attribute*), 46
then_prefixes (*robot.conf.languages.PtBr attribute*), 44
then_prefixes (*robot.conf.languages.Ro attribute*), 61
then_prefixes (*robot.conf.languages.Ru attribute*), 53
then_prefixes (*robot.conf.languages.Sv attribute*), 58
then_prefixes (*robot.conf.languages.Th attribute*), 47
then_prefixes (*robot.conf.languages.Tr attribute*), 57
then_prefixes (*robot.conf.languages.Uk attribute*), 50
then_prefixes (*robot.conf.languages.ZhCn attribute*), 54
then_prefixes (*robot.conf.languages.ZhTw attribute*), 56
time_left() (*robot.running.timeouts.KeywordTimeout method*), 548
time_left() (*robot.running.timeouts.TestTimeout*

method), 548
timed_out() (*robot.running.timeouts.KeywordTimeout method*), 548
timed_out() (*robot.running.timeouts.TestTimeout method*), 548
TimeDeltaConverter (*class in robot.running.arguments.typeconverters*), 536
Timeout (*class in robot.parsing.model.statements*), 368
Timeout (*class in robot.running.timeouts.posix*), 549
Timeout (*class in robot.running.timeouts.windows*), 549
timeout (*robot.errors.BreakLoop attribute*), 618
timeout (*robot.errors.ContinueLoop attribute*), 617
timeout (*robot.errors.ExecutionFailed attribute*), 615
timeout (*robot.errors.ExecutionFailures attribute*), 616
timeout (*robot.errors.ExecutionPassed attribute*), 616
timeout (*robot.errors.ExecutionStatus attribute*), 614
timeout (*robot.errors.HandlerExecutionFailed attribute*), 615
timeout (*robot.errors.PassExecution attribute*), 617
timeout (*robot.errors.ReturnFromKeyword attribute*), 618
timeout (*robot.errors.UserKeywordExecutionFailed attribute*), 616
timeout (*robot.model.keyword.Keyword attribute*), 259
timeout (*robot.model.testcase.TestCase attribute*), 283
TIMEOUT (*robot.parsing.lexer.tokens.END attribute*), 340
TIMEOUT (*robot.parsing.lexer.tokens.EOS attribute*), 337
TIMEOUT (*robot.parsing.lexer.tokens.Token attribute*), 334
timeout (*robot.result.model.Break attribute*), 490
timeout (*robot.result.model.Continue attribute*), 487
timeout (*robot.result.model.For attribute*), 469
timeout (*robot.result.model.ForIteration attribute*), 467
timeout (*robot.result.model.If attribute*), 478
timeout (*robot.result.model.IfBranch attribute*), 476
timeout (*robot.result.model.Keyword attribute*), 493
timeout (*robot.result.model.Return attribute*), 485
timeout (*robot.result.model.TestCase attribute*), 495
timeout (*robot.result.model.Try attribute*), 483
timeout (*robot.result.model.TryBranch attribute*), 480
timeout (*robot.result.model.While attribute*), 474
timeout (*robot.result.model.WhileIteration attribute*), 471
timeout (*robot.result.modeldeprecation.DeprecatedAttributesMixin attribute*), 500
timeout (*robot.running.builder.settings.Defaults attribute*), 543
timeout (*robot.running.builder.settings.TestSettings attribute*), 543
timeout (*robot.running.model.Keyword attribute*), 557
timeout (*robot.running.model.TestCase attribute*), 572
timeout_setting (*robot.conf.languages.Bg attribute*), 60
timeout_setting (*robot.conf.languages.Bs attribute*), 39
timeout_setting (*robot.conf.languages.Cs attribute*), 36
timeout_setting (*robot.conf.languages.De attribute*), 43
timeout_setting (*robot.conf.languages.En attribute*), 35
timeout_setting (*robot.conf.languages.Es attribute*), 51
timeout_setting (*robot.conf.languages.Fi attribute*), 40
timeout_setting (*robot.conf.languages.Fr attribute*), 42
timeout_setting (*robot.conf.languages.Hi attribute*), 64
timeout_setting (*robot.conf.languages.It attribute*), 63
timeout_setting (*robot.conf.languages.Language attribute*), 33
timeout_setting (*robot.conf.languages.Nl attribute*), 37
timeout_setting (*robot.conf.languages.Pl attribute*), 49
timeout_setting (*robot.conf.languages.Pt attribute*), 46
timeout_setting (*robot.conf.languages.PtBr attribute*), 44
timeout_setting (*robot.conf.languages.Ro attribute*), 61
timeout_setting (*robot.conf.languages.Ru attribute*), 53
timeout_setting (*robot.conf.languages.Sv attribute*), 58
timeout_setting (*robot.conf.languages.Th attribute*), 47
timeout_setting (*robot.conf.languages.Tr attribute*), 57
timeout_setting (*robot.conf.languages.Uk attribute*), 50
timeout_setting (*robot.conf.languages.ZhCn attribute*), 54
timeout_setting (*robot.conf.languages.ZhTw attribute*), 56
TimeoutError, 614
TimeoutHandler (*class in robot.result.xmlelementhandlers*), 524
TimeoutHTTPSTransport (*class in robot.libraries.Remote*), 129
TimeoutHTTPTransport (*class in*

robot.libraries.Remote), 129
timestamp (*robot.model.message.Message attribute*), 262
timestamp (*robot.output.loggerhelper.Message attribute*), 311
timestamp (*robot.result.model.Message attribute*), 464
timestamp() (*robot.reporting.jsbuildingcontext.JsBuildingContext method*), 393
title() (*robot.libraries.dialogs_py.InputDialog method*), 179
title() (*robot.libraries.dialogs_py.MessageDialog method*), 165
title() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 207
title() (*robot.libraries.dialogs_py.PassFailDialog method*), 221
title() (*robot.libraries.dialogs_py.SelectionDialog method*), 193
tk_bisque() (*robot.libraries.dialogs_py.InputDialog method*), 179
tk_bisque() (*robot.libraries.dialogs_py.MessageDialog method*), 165
tk_bisque() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 207
tk_bisque() (*robot.libraries.dialogs_py.PassFailDialog method*), 221
tk_bisque() (*robot.libraries.dialogs_py.SelectionDialog method*), 193
tk_focusFollowsMouse() (*robot.libraries.dialogs_py.InputDialog method*), 179
tk_focusFollowsMouse() (*robot.libraries.dialogs_py.MessageDialog method*), 165
tk_focusFollowsMouse() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 207
tk_focusFollowsMouse() (*robot.libraries.dialogs_py.PassFailDialog method*), 221
tk_focusFollowsMouse() (*robot.libraries.dialogs_py.SelectionDialog method*), 193
tk_focusNext() (*robot.libraries.dialogs_py.InputDialog method*), 180
tk_focusNext() (*robot.libraries.dialogs_py.MessageDialog method*), 166
tk_focusNext() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 208
tk_focusNext() (*robot.libraries.dialogs_py.PassFailDialog method*), 222
tk_focusNext() (*robot.libraries.dialogs_py.SelectionDialog method*), 194
tk_focusPrev() (*robot.libraries.dialogs_py.InputDialog method*), 180
tk_focusPrev() (*robot.libraries.dialogs_py.MessageDialog method*), 166
tk_focusPrev() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 208
tk_focusPrev() (*robot.libraries.dialogs_py.PassFailDialog method*), 222
tk_focusPrev() (*robot.libraries.dialogs_py.SelectionDialog method*), 194
tk_setPalette() (*robot.libraries.dialogs_py.InputDialog method*), 180
tk_setPalette() (*robot.libraries.dialogs_py.MessageDialog method*), 166
tk_setPalette() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 208
tk_setPalette() (*robot.libraries.dialogs_py.PassFailDialog method*), 222
tk_setPalette() (*robot.libraries.dialogs_py.SelectionDialog method*), 194
tk_strictMotif() (*robot.libraries.dialogs_py.InputDialog method*), 180
tk_strictMotif() (*robot.libraries.dialogs_py.MessageDialog method*), 166
tk_strictMotif() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 208
tk_strictMotif() (*robot.libraries.dialogs_py.PassFailDialog method*), 222
tk_strictMotif() (*robot.libraries.dialogs_py.SelectionDialog method*), 194
tkraise() (*robot.libraries.dialogs_py.InputDialog method*), 180
tkraise() (*robot.libraries.dialogs_py.MessageDialog method*), 166
tkraise() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 208
tkraise() (*robot.libraries.dialogs_py.PassFailDialog method*), 222
tkraise() (*robot.libraries.dialogs_py.SelectionDialog method*), 194
to_bytes() (*robot.reporting.stringcache.StringIndex method*), 402
to_dictionary() (*robot.libdocpkg.datatypes.EnumMember method*), 70
to_dictionary() (*robot.libdocpkg.datatypes.TypedDictItem method*), 70
to_dictionary() (*robot.libdocpkg.datatypes.TypeDoc method*), 70
to_dictionary() (*robot.libdocpkg.model.KeywordDoc method*), 72
to_dictionary() (*robot.libdocpkg.model.LibraryDoc method*), 72
to_json() (*robot.libdocpkg.model.LibraryDoc method*), 72
Token (*class in robot.parsing.lexer.tokens*), 333

token_type (*robot.parsing.lexer.statementlexers.BreakLexer attribute*), 333
token_type (*robot.parsing.lexer.statementlexers.CommentLexer attribute*), 329
token_type (*robot.parsing.lexer.statementlexers.CommentSectionHeaderLexer attribute*), 329
token_type (*robot.parsing.lexer.statementlexers.ContinueLexer attribute*), 333
token_type (*robot.parsing.lexer.statementlexers.ElseHeaderLexer attribute*), 331
token_type (*robot.parsing.lexer.statementlexers.ElseIfHeaderLexer attribute*), 331
token_type (*robot.parsing.lexer.statementlexers.EndLexer attribute*), 332
token_type (*robot.parsing.lexer.statementlexers.ErrorSectionHeaderLexer attribute*), 329
token_type (*robot.parsing.lexer.statementlexers.ExceptHeaderLexer attribute*), 332
token_type (*robot.parsing.lexer.statementlexers.FinallyHeaderLexer attribute*), 332
token_type (*robot.parsing.lexer.statementlexers.ForHeaderLexer attribute*), 331
token_type (*robot.parsing.lexer.statementlexers.IfHeaderLexer attribute*), 331
token_type (*robot.parsing.lexer.statementlexers.ImplicitCommentLexer attribute*), 330
token_type (*robot.parsing.lexer.statementlexers.InlineIfHeaderLexer attribute*), 331
token_type (*robot.parsing.lexer.statementlexers.KeywordCallLexer attribute*), 331
token_type (*robot.parsing.lexer.statementlexers.KeywordSectionHeaderLexer attribute*), 329
token_type (*robot.parsing.lexer.statementlexers.ReturnLexer attribute*), 332
token_type (*robot.parsing.lexer.statementlexers.SectionHeaderLexer attribute*), 328
token_type (*robot.parsing.lexer.statementlexers.SettingLexer attribute*), 330
token_type (*robot.parsing.lexer.statementlexers.SettingSectionHeaderLexer attribute*), 328
token_type (*robot.parsing.lexer.statementlexers.SingleType attribute*), 328
token_type (*robot.parsing.lexer.statementlexers.StatementLexer attribute*), 327
token_type (*robot.parsing.lexer.statementlexers.TaskSectionHeaderLexer attribute*), 329
token_type (*robot.parsing.lexer.statementlexers.TestCaseSectionHeaderLexer attribute*), 329
token_type (*robot.parsing.lexer.statementlexers.TestOrKeywordSettingLexer attribute*), 330
token_type (*robot.parsing.lexer.statementlexers.TryHeaderLexer attribute*), 332
token_type (*robot.parsing.lexer.statementlexers.TypeAndArgumentsLexer attribute*), 328
token_type (*robot.parsing.lexer.statementlexers.VariableLexer attribute*), 330
token_type (*robot.parsing.lexer.statementlexers.VariableSectionHeaderLexer attribute*), 328
token_type (*robot.parsing.lexer.statementlexers.WhileHeaderLexer attribute*), 332
tokenize() (*robot.parsing.lexer.tokenizer.Tokenizer method*), 333
tokenize_variables() (*robot.parsing.lexer.tokens.END method*), 340
tokenize_variables() (*robot.parsing.lexer.tokens.EOS method*), 338
tokenize_variables() (*robot.parsing.lexer.tokens.Token method*), 335
Tokenizer (*class in robot.parsing.lexer.tokenizer*), 333
TOP_LEVEL_NAME (*robot.running.context.ExecutionContexts attribute*), 550
total (*robot.model.statistics.Statistics attribute*), 270
total (*robot.model.stats.CombinedTagStat attribute*), 276
total (*robot.model.stats.Stat attribute*), 275
total (*robot.model.stats.SuiteStat attribute*), 275
total (*robot.model.stats.TagStat attribute*), 276
total (*robot.model.stats.TotalStat attribute*), 275
total (*robot.model.totalstatistics.TotalStatistics attribute*), 288
TotalStat (*class in robot.model.stats*), 275
TotalStatistics (*class in robot.model.totalstatistics*), 288
TotalStatisticsBuilder (*class in robot.model.totalstatistics*), 288
trace() (*robot.libraries.OperatingSystem.OperatingSystem method*), 122
trace() (*class in robot.conf.languages*), 56
trace() (*in module robot.api.logger*), 15
trace() (*in module robot.output.librarylogger*), 305
trace() (*robot.output.filelogger.FileLogger method*), 305
trace() (*robot.output.logger.Logger method*), 309
trace() (*robot.output.loggerhelper.AbstractLogger method*), 309
trace() (*robot.output.output.Output method*), 311
trace() (*robot.utils.importer.NoLogger method*), 602
traceback (*robot.utils.error.ErrorDetails attribute*), 598
TRANSLATE_WITH_GOOGLE (*robot.libraries.dialogs_py.InputDialog attribute*), 180
translate() (*robot.libraries.dialogs_py.MessageDialog method*), 166
translate() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 208

transient() (*robot.libraries.dialogs_py.PassFailDialog method*), 222
transient() (*robot.libraries.dialogs_py.SelectionDialog method*), 194
true_strings (*robot.conf.languages.Bg attribute*), 60
true_strings (*robot.conf.languages.Bs attribute*), 39
true_strings (*robot.conf.languages.Cs attribute*), 36
true_strings (*robot.conf.languages.De attribute*), 43
true_strings (*robot.conf.languages.En attribute*), 35
true_strings (*robot.conf.languages.Es attribute*), 52
true_strings (*robot.conf.languages.Fi attribute*), 40
true_strings (*robot.conf.languages.Fr attribute*), 42
true_strings (*robot.conf.languages.Hi attribute*), 64
true_strings (*robot.conf.languages.It attribute*), 63
true_strings (*robot.conf.languages.Language attribute*), 33
true_strings (*robot.conf.languages.Nl attribute*), 38
true_strings (*robot.conf.languages.Pl attribute*), 49
true_strings (*robot.conf.languages.Pt attribute*), 46
true_strings (*robot.conf.languages.PtBr attribute*), 45
true_strings (*robot.conf.languages.Ro attribute*), 61
true_strings (*robot.conf.languages.Ru attribute*), 53
true_strings (*robot.conf.languages.Sv attribute*), 59
true_strings (*robot.conf.languages.Th attribute*), 48
true_strings (*robot.conf.languages.Tr attribute*), 57
true_strings (*robot.conf.languages.Uk attribute*), 50
true_strings (*robot.conf.languages.ZhCn attribute*), 54
true_strings (*robot.conf.languages.ZhTw attribute*), 56
Try (*class in robot.model.control*), 245
Try (*class in robot.parsing.model.blocks*), 344
Try (*class in robot.result.model*), 480
Try (*class in robot.running.model*), 564
TRY (*robot.model.body.BodyItem attribute*), 228
TRY (*robot.model.control.Break attribute*), 249
TRY (*robot.model.control.Continue attribute*), 248
TRY (*robot.model.control.For attribute*), 239
TRY (*robot.model.control.If attribute*), 243
TRY (*robot.model.control.IfBranch attribute*), 242
TRY (*robot.model.control.Return attribute*), 247
TRY (*robot.model.control.Try attribute*), 246
TRY (*robot.model.control.TryBranch attribute*), 244
TRY (*robot.model.control.While attribute*), 240
TRY (*robot.model.keyword.Keyword attribute*), 261
TRY (*robot.model.message.Message attribute*), 263
TRY (*robot.output.loggerhelper.Message attribute*), 310
TRY (*robot.parsing.lexer.tokens.END attribute*), 340
TRY (*robot.parsing.lexer.tokens.EOS attribute*), 337
TRY (*robot.parsing.lexer.tokens.Token attribute*), 335
TRY (*robot.result.model.Break attribute*), 488
TRY (*robot.result.model.Continue attribute*), 486
TRY (*robot.result.model.For attribute*), 468
TRY (*robot.result.model.ForIteration attribute*), 465
TRY (*robot.result.model.If attribute*), 477
TRY (*robot.result.model.IfBranch attribute*), 475
TRY (*robot.result.model.Keyword attribute*), 491
TRY (*robot.result.model.Message attribute*), 463
TRY (*robot.result.model.Return attribute*), 484
TRY (*robot.result.model.Try attribute*), 481
TRY (*robot.result.model.TryBranch attribute*), 479
TRY (*robot.result.model.While attribute*), 472
TRY (*robot.result.model.WhileIteration attribute*), 470
TRY (*robot.running.model.Break attribute*), 569
TRY (*robot.running.model.Continue attribute*), 568
TRY (*robot.running.model.For attribute*), 558
TRY (*robot.running.model.If attribute*), 562
TRY (*robot.running.model.IfBranch attribute*), 561
TRY (*robot.running.model.Keyword attribute*), 556
TRY (*robot.running.model.Return attribute*), 566
TRY (*robot.running.model.Try attribute*), 565
TRY (*robot.running.model.TryBranch attribute*), 564
TRY (*robot.running.model.While attribute*), 559
try_branch (*robot.model.control.Try attribute*), 245
try_branch (*robot.result.model.Try attribute*), 483
try_branch (*robot.running.model.Try attribute*), 566
try_class (*robot.model.body.BaseBody attribute*), 229
try_class (*robot.model.body.Body attribute*), 232
try_class (*robot.model.body.Branches attribute*), 234
try_class (*robot.result.model.Body attribute*), 459
try_class (*robot.result.model.Branches attribute*), 461
try_class (*robot.result.model.Iterations attribute*), 462
try_class (*robot.running.model.Body attribute*), 555
TRY_EXCEPT_ROOT (*robot.model.body.BodyItem attribute*), 228
TRY_EXCEPT_ROOT (*robot.model.control.Break attribute*), 249
TRY_EXCEPT_ROOT (*robot.model.control.Continue attribute*), 248
TRY_EXCEPT_ROOT (*robot.model.control.For attribute*), 239
TRY_EXCEPT_ROOT (*robot.model.control.If attribute*), 243
TRY_EXCEPT_ROOT (*robot.model.control.IfBranch attribute*), 242
TRY_EXCEPT_ROOT (*robot.model.control.Return attribute*), 247
TRY_EXCEPT_ROOT (*robot.model.control.Try attribute*), 246
TRY_EXCEPT_ROOT (*robot.model.control.TryBranch attribute*), 244
TRY_EXCEPT_ROOT (*robot.model.control.While attribute*), 240

TRY_EXCEPT_ROOT (*robot.model.keyword.Keyword attribute*), 261
TRY_EXCEPT_ROOT (*robot.model.message.Message attribute*), 263
TRY_EXCEPT_ROOT (*robot.output.loggerhelper.Message attribute*), 310
TRY_EXCEPT_ROOT (*robot.result.model.Break attribute*), 488
TRY_EXCEPT_ROOT (*robot.result.model.Continue attribute*), 486
TRY_EXCEPT_ROOT (*robot.result.model.For attribute*), 468
TRY_EXCEPT_ROOT (*robot.result.model.ForIteration attribute*), 465
TRY_EXCEPT_ROOT (*robot.result.model.If attribute*), 477
TRY_EXCEPT_ROOT (*robot.result.model.IfBranch attribute*), 475
TRY_EXCEPT_ROOT (*robot.result.model.Keyword attribute*), 491
TRY_EXCEPT_ROOT (*robot.result.model.Message attribute*), 463
TRY_EXCEPT_ROOT (*robot.result.model.Return attribute*), 484
TRY_EXCEPT_ROOT (*robot.result.model.Try attribute*), 481
TRY_EXCEPT_ROOT (*robot.result.model.TryBranch attribute*), 479
TRY_EXCEPT_ROOT (*robot.result.model.While attribute*), 472
TRY_EXCEPT_ROOT (*robot.result.model.WhileIteration attribute*), 470
TRY_EXCEPT_ROOT (*robot.running.model.Break attribute*), 569
TRY_EXCEPT_ROOT (*robot.running.model.Continue attribute*), 568
TRY_EXCEPT_ROOT (*robot.running.model.For attribute*), 558
TRY_EXCEPT_ROOT (*robot.running.model.If attribute*), 562
TRY_EXCEPT_ROOT (*robot.running.model.IfBranch attribute*), 561
TRY_EXCEPT_ROOT (*robot.running.model.Keyword attribute*), 556
TRY_EXCEPT_ROOT (*robot.running.model.Return attribute*), 566
TRY_EXCEPT_ROOT (*robot.running.model.Try attribute*), 565
TRY_EXCEPT_ROOT (*robot.running.model.TryBranch attribute*), 564
TRY_EXCEPT_ROOT (*robot.running.model.While attribute*), 559
TryBranch (*class in robot.model.control*), 244
TryBranch (*class in robot.result.model*), 478
TryBranch (*class in robot.running.model*), 563
TryBuilder (*class in robot.running.builder.transformers*), 547
TryHandler (*class in robot.result.xmlelementhandlers*), 521
TryHeader (*class in robot.parsing.model.statements*), 378
TryHeaderLexer (*class in robot.parsing.lexer.statementlexers*), 331
TryLexer (*class in robot.parsing.lexer.blocklexers*), 323
TryParser (*class in robot.parsing.parser.blockparsers*), 390
TryRunner (*class in robot.running.bodyrunner*), 550
TupleConverter (*class in robot.running.arguments.typeconverters*), 538
TupleListDumper (*class in robot.htmldata.jsonwriter*), 68
type (*robot.libdocpkg.robotbuilder.ResourceDocBuilder attribute*), 72
type (*robot.libdocpkg.robotbuilder.SuiteDocBuilder attribute*), 72
type (*robot.model.body.BodyItem attribute*), 228
type (*robot.model.control.Break attribute*), 250
type (*robot.model.control.Continue attribute*), 248
type (*robot.model.control.For attribute*), 238
type (*robot.model.control.If attribute*), 242
type (*robot.model.control.IfBranch attribute*), 241
type (*robot.model.control.Return attribute*), 246
type (*robot.model.control.Try attribute*), 245
type (*robot.model.control.TryBranch attribute*), 244
type (*robot.model.control.While attribute*), 240
type (*robot.model.keyword.Keyword attribute*), 259
type (*robot.model.message.Message attribute*), 262
type (*robot.model.stats.CombinedTagStat attribute*), 276
type (*robot.model.stats.SuiteStat attribute*), 275
type (*robot.model.stats.TagStat attribute*), 276
type (*robot.model.stats.TotalStat attribute*), 275
type (*robot.output.loggerhelper.Message attribute*), 311
type (*robot.parsing.lexer.tokens.END attribute*), 340
type (*robot.parsing.lexer.tokens.EOS attribute*), 338
type (*robot.parsing.lexer.tokens.Token attribute*), 335
type (*robot.parsing.model.blocks.If attribute*), 343
type (*robot.parsing.model.blocks.Try attribute*), 344
type (*robot.parsing.model.statements.Arguments attribute*), 368
type (*robot.parsing.model.statements.Break attribute*), 384
type (*robot.parsing.model.statements.Comment attribute*), 385
type (*robot.parsing.model.statements.Config attribute*), 386

type (*robot.parsing.model.statements.Continue attribute*), 383
type (*robot.parsing.model.statements.DefaultTags attribute*), 355
type (*robot.parsing.model.statements.Documentation attribute*), 353
type (*robot.parsing.model.statements.DocumentationOrMetadata attribute*), 347
type (*robot.parsing.model.statements.ElseHeader attribute*), 376
type (*robot.parsing.model.statements.ElseIfHeader attribute*), 375
type (*robot.parsing.model.statements.EmptyLine attribute*), 388
type (*robot.parsing.model.statements.End attribute*), 380
type (*robot.parsing.model.statements.Error attribute*), 387
type (*robot.parsing.model.statements.ExceptHeader attribute*), 378
type (*robot.parsing.model.statements.FinallyHeader attribute*), 379
type (*robot.parsing.model.statements.Fixture attribute*), 349
type (*robot.parsing.model.statements.ForceTags attribute*), 354
type (*robot.parsing.model.statements.ForHeader attribute*), 372
type (*robot.parsing.model.statements.IfElseHeader attribute*), 373
type (*robot.parsing.model.statements.IfHeader attribute*), 373
type (*robot.parsing.model.statements.InlineIfHeader attribute*), 374
type (*robot.parsing.model.statements.KeywordCall attribute*), 370
type (*robot.parsing.model.statements.KeywordName attribute*), 363
type (*robot.parsing.model.statements.KeywordTags attribute*), 356
type (*robot.parsing.model.statements.LibraryImport attribute*), 350
type (*robot.parsing.model.statements.LoopControl attribute*), 383
type (*robot.parsing.model.statements.Metadata attribute*), 354
type (*robot.parsing.model.statements.MultiValue attribute*), 349
type (*robot.parsing.model.statements.NoArgumentHeader attribute*), 377
type (*robot.parsing.model.statements.ResourceImport attribute*), 351
type (*robot.parsing.model.statements.Return attribute*), 369
type (*robot.parsing.model.statements.ReturnStatement attribute*), 382
type (*robot.parsing.model.statements.SectionHeader attribute*), 350
type (*robot.parsing.model.statements.Setup attribute*), 364
type (*robot.parsing.model.statements.SingleValue attribute*), 348
type (*robot.parsing.model.statements.Statement attribute*), 345
type (*robot.parsing.model.statements.SuiteSetup attribute*), 357
type (*robot.parsing.model.statements.SuiteTeardown attribute*), 358
type (*robot.parsing.model.statements.Tags attribute*), 366
type (*robot.parsing.model.statements.Teardown attribute*), 365
type (*robot.parsing.model.statements.Template attribute*), 367
type (*robot.parsing.model.statements.TemplateArguments attribute*), 371
type (*robot.parsing.model.statements.TestCaseName attribute*), 363
type (*robot.parsing.model.statements.TestSetup attribute*), 359
type (*robot.parsing.model.statements.TestTeardown attribute*), 359
type (*robot.parsing.model.statements.TestTemplate attribute*), 360
type (*robot.parsing.model.statements.TestTimeout attribute*), 361
type (*robot.parsing.model.statements.Timeout attribute*), 368
type (*robot.parsing.model.statements.TryHeader attribute*), 378
type (*robot.parsing.model.statements.Variable attribute*), 362
type (*robot.parsing.model.statements.VariablesImport attribute*), 352
type (*robot.parsing.model.statements.WhileHeader attribute*), 381
type (*robot.result.model.Break attribute*), 490
type (*robot.result.model.Continue attribute*), 487
type (*robot.result.model.For attribute*), 469
type (*robot.result.model.ForIteration attribute*), 465
type (*robot.result.model.If attribute*), 478
type (*robot.result.model.IfBranch attribute*), 476
type (*robot.result.model.Keyword attribute*), 493
type (*robot.result.model.Message attribute*), 464
type (*robot.result.model.Return attribute*), 485
type (*robot.result.model.Try attribute*), 483
type (*robot.result.model.TryBranch attribute*), 480
type (*robot.result.model.While attribute*), 474

type (*robot.result.model.WhileIteration attribute*), 469
type (*robot.running.arguments.typeconverters.BooleanConverter attribute*), 533
type (*robot.running.arguments.typeconverters.ByteArrayConverter attribute*), 535
type (*robot.running.arguments.typeconverters.BytesConverter attribute*), 534
type (*robot.running.arguments.typeconverters.CombinedConverter attribute*), 540
type (*robot.running.arguments.typeconverters.CustomConverter attribute*), 541
type (*robot.running.arguments.typeconverters.DateConverter attribute*), 536
type (*robot.running.arguments.typeconverters.DateTimeConverter attribute*), 535
type (*robot.running.arguments.typeconverters.DecimalConverter attribute*), 534
type (*robot.running.arguments.typeconverters.DictionaryConverter attribute*), 539
type (*robot.running.arguments.typeconverters.EnumConverter attribute*), 532
type (*robot.running.arguments.typeconverters.FloatConverter attribute*), 534
type (*robot.running.arguments.typeconverters.FrozenSetConverter attribute*), 540
type (*robot.running.arguments.typeconverters.IntegerConverter attribute*), 533
type (*robot.running.arguments.typeconverters.ListConverter attribute*), 537
type (*robot.running.arguments.typeconverters.NoneConverter attribute*), 537
type (*robot.running.arguments.typeconverters.PathConverter attribute*), 537
type (*robot.running.arguments.typeconverters.SetConverter attribute*), 539
type (*robot.running.arguments.typeconverters.StringConverter attribute*), 532
type (*robot.running.arguments.typeconverters.TimeDeltaConverter attribute*), 536
type (*robot.running.arguments.typeconverters.TupleConverter attribute*), 538
type (*robot.running.arguments.typeconverters.TypeConverter attribute*), 531
type (*robot.running.arguments.typeconverters.TypedDictConverter attribute*), 538
type (*robot.running.model.Break attribute*), 570
type (*robot.running.model.Continue attribute*), 568
type (*robot.running.model.For attribute*), 559
type (*robot.running.model.If attribute*), 563
type (*robot.running.model.IfBranch attribute*), 561
type (*robot.running.model.Keyword attribute*), 557
type (*robot.running.model.Return attribute*), 567
type (*robot.running.model.Try attribute*), 566
type (*robot.running.model.TryBranch attribute*), 564
type (*robot.running.model.While attribute*), 560
type (*robot.running.timeouts.KeywordTimeout attribute*), 548
type (*robot.running.timeouts.TestTimeout attribute*), 548
type_docs (*robot.libdocpkg.model.LibraryDoc attribute*), 71
type_name (*robot.running.arguments.typeconverters.BooleanConverter attribute*), 533
type_name (*robot.running.arguments.typeconverters.ByteArrayConverter attribute*), 535
type_name (*robot.running.arguments.typeconverters.BytesConverter attribute*), 535
type_name (*robot.running.arguments.typeconverters.CombinedConverter attribute*), 540
type_name (*robot.running.arguments.typeconverters.CustomConverter attribute*), 541
type_name (*robot.running.arguments.typeconverters.DateConverter attribute*), 536
type_name (*robot.running.arguments.typeconverters.DateTimeConverter attribute*), 535
type_name (*robot.running.arguments.typeconverters.DecimalConverter attribute*), 534
type_name (*robot.running.arguments.typeconverters.DictionaryConverter attribute*), 539
type_name (*robot.running.arguments.typeconverters.EnumConverter attribute*), 532
type_name (*robot.running.arguments.typeconverters.FloatConverter attribute*), 534
type_name (*robot.running.arguments.typeconverters.FrozenSetConverter attribute*), 540
type_name (*robot.running.arguments.typeconverters.IntegerConverter attribute*), 533
type_name (*robot.running.arguments.typeconverters.ListConverter attribute*), 538
type_name (*robot.running.arguments.typeconverters.NoneConverter attribute*), 537
type_name (*robot.running.arguments.typeconverters.PathConverter attribute*), 537
type_name (*robot.running.arguments.typeconverters.SetConverter attribute*), 539
type_name (*robot.running.arguments.typeconverters.StringConverter attribute*), 532
type_name (*robot.running.arguments.typeconverters.TimeDeltaConverter attribute*), 536
type_name (*robot.running.arguments.typeconverters.TupleConverter attribute*), 538
type_name (*robot.running.arguments.typeconverters.TypeConverter attribute*), 531
type_name (*robot.running.arguments.typeconverters.TypedDictConverter attribute*), 538
TypeAndArguments (*class in robot.parsing.lexer.statementlexers*), 328
TypeConverter (*class in*

robot.running.arguments.typeconverters), 531
TYPED_DICT (*robot.libdocpkg.datatypes.TypeDoc attribute*), 70
TypedDictConverter (*class in robot.running.arguments.typeconverters*), 538
TypedDictItem (*class in robot.libdocpkg.datatypes*), 70
TypeDoc (*class in robot.libdocpkg.datatypes*), 70
types (*robot.running.arguments.argumentspec.ArgInfo attribute*), 530
types (*robot.running.arguments.argumentspec.ArgumentSpec attribute*), 530
types_reprs (*robot.running.arguments.argumentspec.ArgInfo attribute*), 530
TypeValidator (*class in robot.running.arguments.typevalidator*), 541

U

Uk (*class in robot.conf.languages*), 49
unbind() (*robot.libraries.dialogs_py.InputDialog method*), 180
unbind() (*robot.libraries.dialogs_py.MessageDialog method*), 166
unbind() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 208
unbind() (*robot.libraries.dialogs_py.PassFailDialog method*), 222
unbind() (*robot.libraries.dialogs_py.SelectionDialog method*), 194
unbind_all() (*robot.libraries.dialogs_py.InputDialog method*), 180
unbind_all() (*robot.libraries.dialogs_py.MessageDialog method*), 166
unbind_all() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 208
unbind_all() (*robot.libraries.dialogs_py.PassFailDialog method*), 222
unbind_all() (*robot.libraries.dialogs_py.SelectionDialog method*), 194
unbind_class() (*robot.libraries.dialogs_py.InputDialog method*), 180
unbind_class() (*robot.libraries.dialogs_py.MessageDialog method*), 166
unbind_class() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 208
unbind_class() (*robot.libraries.dialogs_py.PassFailDialog method*), 222
unbind_class() (*robot.libraries.dialogs_py.SelectionDialog method*), 194
unescape() (*robot.utils.escaping.Unescaper method*), 598
unescape_variable_syntax() (*in module robot.variables.search*), 611
Unescaper (*class in robot.utils.escaping*), 598
unhandled_tokens (*robot.parsing.parser.blockparsers.BlockParser attribute*), 389
unhandled_tokens (*robot.parsing.parser.blockparsers.ForParser attribute*), 390
unhandled_tokens (*robot.parsing.parser.blockparsers.IfParser attribute*), 390
unhandled_tokens (*robot.parsing.parser.blockparsers.KeywordParser attribute*), 389
unhandled_tokens (*robot.parsing.parser.blockparsers.NestedBlockPar attribute*), 390
unhandled_tokens (*robot.parsing.parser.blockparsers.TestCaseParser attribute*), 389
unhandled_tokens (*robot.parsing.parser.blockparsers.TryParser attribute*), 390
unhandled_tokens (*robot.parsing.parser.blockparsers.WhileParser attribute*), 390
unregister() (*robot.output.listenermethods.LibraryListenerMethods method*), 307
unregister() (*robot.output.listeners.LibraryListeners method*), 307
unregister_console_logger() (*robot.output.logger.Logger method*), 308
unregister_logger() (*robot.output.logger.Logger method*), 308
unregister_xml_logger() (*robot.output.logger.Logger method*), 308
unstrip() (*robot.libraries.XML.NameSpaceStripper method*), 157
update() (*robot.libraries.dialogs_py.InputDialog method*), 180
update() (*robot.libraries.dialogs_py.MessageDialog method*), 166
update() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 208
update() (*robot.libraries.dialogs_py.PassFailDialog method*), 222
update() (*robot.libraries.dialogs_py.SelectionDialog method*), 194
update() (*robot.model.metadata.Metadata method*), 265
update() (*robot.utils.dotdict.DotDict method*), 597
update() (*robot.utils.normalizing.NormalizedDict method*), 605
update() (*robot.variables.evaluation.EvaluationNamespace method*), 607
update() (*robot.variables.scopes.GlobalVariables method*), 610
update() (*robot.variables.scopes.SetVariables method*), 610
update() (*robot.variables.store.VariableStore method*), 611

update() (*robot.variables.variables.Variables method*), 613
update_idletasks() (*robot.libraries.dialogs_py.InputDialog method*), 180
update_idletasks() (*robot.libraries.dialogs_py.MessageDialog method*), 166
update_idletasks() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 208
update_idletasks() (*robot.libraries.dialogs_py.PassFailDialog method*), 222
update_idletasks() (*robot.libraries.dialogs_py.SelectionDialog method*), 194
usage (*robot.reporting.logreportwriters.LogWriter attribute*), 395
usage (*robot.reporting.logreportwriters.ReportWriter attribute*), 395
user_agent (*robot.libraries.Remote.TimeoutHTTPSTransport attribute*), 130
user_agent (*robot.libraries.Remote.TimeoutHTTPTransport attribute*), 129
UserErrorHandler (*class in robot.running.usererrorhandler*), 589
UserKeyword (*class in robot.running.model*), 576
UserKeywordArgumentParser (*class in robot.running.arguments.argumentparser*), 529
UserKeywordExecutionFailed, 616
UserKeywordHandler (*class in robot.running.userkeyword*), 589
UserKeywordRunner (*class in robot.running.userkeywordrunner*), 590
UserLibrary (*class in robot.running.userkeyword*), 589

V

validate() (*robot.libdoc.LibDoc method*), 619
validate() (*robot.parsing.model.blocks.Block method*), 340
validate() (*robot.parsing.model.blocks.CommentSection method*), 342
validate() (*robot.parsing.model.blocks.File method*), 341
validate() (*robot.parsing.model.blocks.For method*), 344
validate() (*robot.parsing.model.blocks.HeaderAndBody method*), 341
validate() (*robot.parsing.model.blocks.If method*), 343
validate() (*robot.parsing.model.blocks.Keyword method*), 343
validate() (*robot.parsing.model.blocks.KeywordSection method*), 342
validate() (*robot.parsing.model.blocks.Section method*), 341
validate() (*robot.parsing.model.blocks.SettingSection method*), 341
validate() (*robot.parsing.model.blocks.TestCase method*), 343
validate() (*robot.parsing.model.blocks.TestCaseSection method*), 342
validate() (*robot.parsing.model.blocks.Try method*), 344
validate() (*robot.parsing.model.blocks.VariableSection method*), 342
validate() (*robot.parsing.model.blocks.While method*), 344
validate() (*robot.parsing.model.statements.Arguments method*), 369
validate() (*robot.parsing.model.statements.Break method*), 385
validate() (*robot.parsing.model.statements.Comment method*), 386
validate() (*robot.parsing.model.statements.Config method*), 387
validate() (*robot.parsing.model.statements.Continue method*), 384
validate() (*robot.parsing.model.statements.DefaultTags method*), 356
validate() (*robot.parsing.model.statements.Documentation method*), 354
validate() (*robot.parsing.model.statements.DocumentationOrMetadata method*), 347
validate() (*robot.parsing.model.statements.ElseHeader method*), 376
validate() (*robot.parsing.model.statements.ElseIfHeader method*), 376
validate() (*robot.parsing.model.statements.EmptyLine method*), 388
validate() (*robot.parsing.model.statements.End method*), 381
validate() (*robot.parsing.model.statements.Error method*), 387
validate() (*robot.parsing.model.statements.ExceptHeader method*), 379
validate() (*robot.parsing.model.statements.FinallyHeader method*), 380
validate() (*robot.parsing.model.statements.Fixture method*), 349
validate() (*robot.parsing.model.statements.ForceTags method*), 355
validate() (*robot.parsing.model.statements.ForHeader method*), 372

validate() (*robot.parsing.model.statements.IfElseHeader method*), 373
validate() (*robot.parsing.model.statements.IfHeader method*), 374
validate() (*robot.parsing.model.statements.InlineIfHeader method*), 375
validate() (*robot.parsing.model.statements.KeywordCall method*), 371
validate() (*robot.parsing.model.statements.KeywordName method*), 364
validate() (*robot.parsing.model.statements.KeywordTags method*), 357
validate() (*robot.parsing.model.statements.LibraryImport method*), 351
validate() (*robot.parsing.model.statements.LoopControl method*), 383
validate() (*robot.parsing.model.statements.Metadata method*), 354
validate() (*robot.parsing.model.statements.MultiValue method*), 349
validate() (*robot.parsing.model.statements.NoArgumentHeader method*), 377
validate() (*robot.parsing.model.statements.ResourceImport method*), 352
validate() (*robot.parsing.model.statements.Return method*), 370
validate() (*robot.parsing.model.statements.ReturnStatement method*), 382
validate() (*robot.parsing.model.statements.SectionHeader method*), 350
validate() (*robot.parsing.model.statements.Setup method*), 365
validate() (*robot.parsing.model.statements.SingleValue method*), 348
validate() (*robot.parsing.model.statements.Statement method*), 346
validate() (*robot.parsing.model.statements.SuiteSetup method*), 358
validate() (*robot.parsing.model.statements.SuiteTeardown method*), 359
validate() (*robot.parsing.model.statements.Tags method*), 367
validate() (*robot.parsing.model.statements.Teardown method*), 366
validate() (*robot.parsing.model.statements.Template method*), 367
validate() (*robot.parsing.model.statements.TemplateArguments method*), 372
validate() (*robot.parsing.model.statements.TestCaseName method*), 363
validate() (*robot.parsing.model.statements.TestSetup method*), 359
validate() (*robot.parsing.model.statements.TestTeardown method*), 360
validate() (*robot.parsing.model.statements.TestTemplate method*), 361
validate() (*robot.parsing.model.statements.TestTimeout method*), 362
validate() (*robot.parsing.model.statements.Timeout method*), 368
validate() (*robot.parsing.model.statements.TryHeader method*), 378
validate() (*robot.parsing.model.statements.Variable method*), 362
validate() (*robot.parsing.model.statements.VariablesImport method*), 353
validate() (*robot.parsing.model.statements.WhileHeader method*), 381
validate() (*robot.rebot.Rebot method*), 621
validate() (*robot.run.RobotFramework method*), 622
validate() (*robot.running.arguments.argumentvalidator.ArgumentValid method*), 531
validate() (*robot.running.arguments.embedded.EmbeddedArguments method*), 531
validate() (*robot.running.arguments.typevalidator.TypeValidator method*), 541
validate() (*robot.testdoc.TestDoc method*), 624
validate() (*robot.utils.application.Application method*), 591
validate() (*robot.variables.assigner.AssignmentValidator method*), 606
validate_assignment() (*robot.variables.assigner.VariableAssignment method*), 606
validate_command() (*robot.libdocpkg.consoleviewer.ConsoleViewer class method*), 69
validate_flatten_keyword() (*in module robot.result.flattenkeywordmatcher*), 415
validate_model() (*robot.parsing.model.blocks.Block method*), 340
validate_model() (*robot.parsing.model.blocks.CommentSection method*), 342
validate_model() (*robot.parsing.model.blocks.File method*), 341
validate_model() (*robot.parsing.model.blocks.For method*), 344
validate_model() (*robot.parsing.model.blocks.HeaderAndBody method*), 341
validate_model() (*robot.parsing.model.blocks.If method*), 343
validate_model() (*robot.parsing.model.blocks.Keyword method*), 343
validate_model() (*robot.parsing.model.blocks.KeywordSection method*), 342
validate_model() (*robot.parsing.model.blocks.Section method*), 341
validate_model() (*robot.parsing.model.blocks.SettingSection*

method), 341
validate_model() (*robot.parsing.model.blocks.TestCase method*), 343
validate_model() (*robot.parsing.model.blocks.TestCaseSection method*), 342
validate_model() (*robot.parsing.model.blocks.Try method*), 344
validate_model() (*robot.parsing.model.blocks.VariableSection method*), 342
validate_model() (*robot.parsing.model.blocks.While method*), 344
validate_type_dict() (*robot.running.arguments.typevalidator.TypeValidator method*), 541
ValidationContext (*class in robot.parsing.model.blocks*), 345
value (*robot.parsing.lexer.tokens.END attribute*), 340
value (*robot.parsing.lexer.tokens.EOS attribute*), 338
value (*robot.parsing.lexer.tokens.Token attribute*), 335
value (*robot.parsing.model.statements.Documentation attribute*), 353
value (*robot.parsing.model.statements.Metadata attribute*), 354
value (*robot.parsing.model.statements.SingleValue attribute*), 347
value (*robot.parsing.model.statements.Template attribute*), 367
value (*robot.parsing.model.statements.TestTemplate attribute*), 361
value (*robot.parsing.model.statements.TestTimeout attribute*), 362
value (*robot.parsing.model.statements.Timeout attribute*), 368
value (*robot.parsing.model.statements.Variable attribute*), 362
value_types (*robot.running.arguments.typeconverters.BooleanConverter attribute*), 533
value_types (*robot.running.arguments.typeconverters.ByteArrayConverter attribute*), 535
value_types (*robot.running.arguments.typeconverters.BytesConverter attribute*), 535
value_types (*robot.running.arguments.typeconverters.CombinedConverter attribute*), 540
value_types (*robot.running.arguments.typeconverters.CustomConverter attribute*), 541
value_types (*robot.running.arguments.typeconverters.DateConverter attribute*), 536
value_types (*robot.running.arguments.typeconverters.DateTimeConverter attribute*), 535
value_types (*robot.running.arguments.typeconverters.DecimalConverter attribute*), 534
value_types (*robot.running.arguments.typeconverters.DictionaryConverter attribute*), 539
value_types (*robot.running.arguments.typeconverters.EnumConverter attribute*), 532
value_types (*robot.running.arguments.typeconverters.FloatConverter attribute*), 534
value_types (*robot.running.arguments.typeconverters.FrozenSetConverter attribute*), 540
value_types (*robot.running.arguments.typeconverters.IntegerConverter attribute*), 533
value_types (*robot.running.arguments.typeconverters.ListConverter attribute*), 537
value_types (*robot.running.arguments.typeconverters.NoneConverter attribute*), 537
value_types (*robot.running.arguments.typeconverters.PathConverter attribute*), 537
value_types (*robot.running.arguments.typeconverters.SetConverter attribute*), 539
value_types (*robot.running.arguments.typeconverters.StringConverter attribute*), 532
value_types (*robot.running.arguments.typeconverters.TimeDeltaConverter attribute*), 536
value_types (*robot.running.arguments.typeconverters.TupleConverter attribute*), 538
value_types (*robot.running.arguments.typeconverters.TypeConverter attribute*), 532
value_types (*robot.running.arguments.typeconverters.TypedDictConverter attribute*), 538
ValueHandler (*class in robot.result.xmlelementhandlers*), 526
values (*robot.model.control.For attribute*), 238
values (*robot.model.control.Return attribute*), 246
values (*robot.parsing.model.blocks.For attribute*), 344
values (*robot.parsing.model.statements.Arguments attribute*), 369
values (*robot.parsing.model.statements.Break attribute*), 385
values (*robot.parsing.model.statements.Continue attribute*), 384
values (*robot.parsing.model.statements.DefaultTags attribute*), 356
values (*robot.parsing.model.statements.End attribute*), 381
values (*robot.parsing.model.statements.FinallyHeader attribute*), 380
values (*robot.parsing.model.statements.ForceTags attribute*), 355
values (*robot.parsing.model.statements.ForHeader attribute*), 372
values (*robot.parsing.model.statements.KeywordTags attribute*), 357
values (*robot.parsing.model.statements.LoopControl attribute*), 383
values (*robot.parsing.model.statements.MultiValue attribute*), 348
values (*robot.parsing.model.statements.NoArgumentHeader attribute*), 377

values (*robot.parsing.model.statements.Return attribute*), 370
values (*robot.parsing.model.statements.ReturnStatement attribute*), 382
values (*robot.parsing.model.statements.Tags attribute*), 367
values (*robot.parsing.model.statements.TryHeader attribute*), 378
values (*robot.result.model.For attribute*), 469
values (*robot.result.model.Return attribute*), 485
values (*robot.running.model.For attribute*), 559
values (*robot.running.model.Return attribute*), 567
values() (*robot.model.metadata.Metadata method*), 265
values() (*robot.running.importer.ImportCache method*), 552
values() (*robot.utils.dotdict.DotDict method*), 597
values() (*robot.utils.normalizing.NormalizedDict method*), 605
values() (*robot.variables.evaluation.EvaluationNamespace method*), 607
VAR_NAMED (*robot.running.arguments.argumentspec.ArgInfo attribute*), 530
VAR_POSITIONAL (*robot.running.arguments.argumentspec.ArgInfo attribute*), 530
VarHandler (*class in robot.result.xmlelementhandlers*), 525
Variable (*class in robot.parsing.model.statements*), 362
Variable (*class in robot.running.model*), 576
variable (*robot.model.control.TryBranch attribute*), 244
VARIABLE (*robot.parsing.lexer.tokens.END attribute*), 340
VARIABLE (*robot.parsing.lexer.tokens.EOS attribute*), 337
VARIABLE (*robot.parsing.lexer.tokens.Token attribute*), 334
variable (*robot.parsing.model.blocks.Try attribute*), 344
variable (*robot.parsing.model.statements.ExceptHeader attribute*), 379
variable (*robot.result.model.TryBranch attribute*), 480
variable (*robot.running.model.TryBranch attribute*), 564
variable_files (*robot.conf.settings.RobotSettings attribute*), 65
VARIABLE_HEADER (*robot.parsing.lexer.tokens.END attribute*), 340
VARIABLE_HEADER (*robot.parsing.lexer.tokens.EOS attribute*), 337
VARIABLE_HEADER (*robot.parsing.lexer.tokens.Token attribute*), 333
variable_not_found() (*in module robot.variables.notfound*), 608
variable_section() (*robot.parsing.lexer.context.FileContext method*), 323
variable_section() (*robot.parsing.lexer.context.InitFileContext method*), 325
variable_section() (*robot.parsing.lexer.context.ResourceFileContext method*), 324
variable_section() (*robot.parsing.lexer.context.TestCaseFileContext method*), 324
variable_should_exist() (*robot.libraries.BuiltIn.BuiltIn method*), 98
variable_should_not_exist() (*robot.libraries.BuiltIn.BuiltIn method*), 98
VariableAssigner (*class in robot.variables.assigner*), 606
VariableAssignment (*class in robot.variables.assigner*), 606
VariableError, 613
VariableFileSetter (*class in robot.variables.filesetter*), 607
VariableFinder (*class in robot.variables.finders*), 608
VariableIterator (*class in robot.variables.search*), 611
VariableLexer (*class in robot.parsing.lexer.statementlexers*), 330
VariableMatch (*class in robot.variables.search*), 611
VariableReplacer (*class in robot.running.arguments.argumentresolver*), 529
VariableReplacer (*class in robot.variables.replacer*), 609
Variables (*class in robot.variables.variables*), 612
variables (*robot.conf.settings.RobotSettings attribute*), 65
variables (*robot.model.control.For attribute*), 238
VARIABLES (*robot.parsing.lexer.tokens.END attribute*), 340
VARIABLES (*robot.parsing.lexer.tokens.EOS attribute*), 337
VARIABLES (*robot.parsing.lexer.tokens.Token attribute*), 334
variables (*robot.parsing.model.blocks.For attribute*), 343
variables (*robot.parsing.model.statements.ForHeader attribute*), 372
variables (*robot.result.model.For attribute*), 469

variables (*robot.result.model.ForIteration attribute*), 465
variables (*robot.running.model.For attribute*), 559
variables (*robot.running.model.ResourceFile attribute*), 576
variables() (*robot.running.model.Imports method*), 577
variables_header (*robot.conf.languages.Bg attribute*), 59
variables_header (*robot.conf.languages.Bs attribute*), 38
variables_header (*robot.conf.languages.Cs attribute*), 35
variables_header (*robot.conf.languages.De attribute*), 42
variables_header (*robot.conf.languages.En attribute*), 34
variables_header (*robot.conf.languages.Es attribute*), 51
variables_header (*robot.conf.languages.Fi attribute*), 39
variables_header (*robot.conf.languages.Fr attribute*), 41
variables_header (*robot.conf.languages.Hi attribute*), 63
variables_header (*robot.conf.languages.It attribute*), 62
variables_header (*robot.conf.languages.Language attribute*), 32
variables_header (*robot.conf.languages.Nl attribute*), 37
variables_header (*robot.conf.languages.Pl attribute*), 48
variables_header (*robot.conf.languages.Pt attribute*), 45
variables_header (*robot.conf.languages.PtBr attribute*), 44
variables_header (*robot.conf.languages.Ro attribute*), 60
variables_header (*robot.conf.languages.Ru attribute*), 52
variables_header (*robot.conf.languages.Sv attribute*), 58
variables_header (*robot.conf.languages.Th attribute*), 46
variables_header (*robot.conf.languages.Tr attribute*), 56
variables_header (*robot.conf.languages.Uk attribute*), 49
variables_header (*robot.conf.languages.ZhCn attribute*), 53
variables_header (*robot.conf.languages.ZhTw attribute*), 55
variables_setting (*robot.conf.languages.Bg attribute*), 59
variables_setting (*robot.conf.languages.Bs attribute*), 38
variables_setting (*robot.conf.languages.Cs attribute*), 35
variables_setting (*robot.conf.languages.De attribute*), 42
variables_setting (*robot.conf.languages.En attribute*), 34
variables_setting (*robot.conf.languages.Es attribute*), 51
variables_setting (*robot.conf.languages.Fi attribute*), 40
variables_setting (*robot.conf.languages.Fr attribute*), 41
variables_setting (*robot.conf.languages.Hi attribute*), 63
variables_setting (*robot.conf.languages.It attribute*), 62
variables_setting (*robot.conf.languages.Language attribute*), 32
variables_setting (*robot.conf.languages.Nl attribute*), 37
variables_setting (*robot.conf.languages.Pl attribute*), 48
variables_setting (*robot.conf.languages.Pt attribute*), 45
variables_setting (*robot.conf.languages.PtBr attribute*), 44
variables_setting (*robot.conf.languages.Ro attribute*), 61
variables_setting (*robot.conf.languages.Ru attribute*), 52
variables_setting (*robot.conf.languages.Sv attribute*), 58
variables_setting (*robot.conf.languages.Th attribute*), 47
variables_setting (*robot.conf.languages.Tr attribute*), 56
variables_setting (*robot.conf.languages.Uk attribute*), 49
variables_setting (*robot.conf.languages.ZhCn attribute*), 54
variables_setting (*robot.conf.languages.ZhTw attribute*), 55
VariableScopes (*class in robot.variables.scopes*), 609
VariableSection (*class in robot.parsing.model.blocks*), 341
VariableSectionHeaderLexer (*class in robot.parsing.lexer.statementlexers*), 328
VariableSectionLexer (*class in robot.parsing.lexer.blocklexers*), 319

VariableSectionParser (*class in robot.parsing.parser.fileparser*), 391
VariablesImport (*class in robot.parsing.model.statements*), 352
VariableStore (*class in robot.variables.store*), 611
VariableTableSetter (*class in robot.variables.tablesetter*), 612
VariableTableValue() (*in module robot.variables.tablesetter*), 612
VariableTableValueBase (*class in robot.variables.tablesetter*), 612
VerboseOutput (*class in robot.output.console.verbose*), 304
VerboseWriter (*class in robot.output.console.verbose*), 304
version() (*robot.libdocpkg.consoleviewer.ConsoleViewer method*), 70
view() (*robot.libdocpkg.consoleviewer.ConsoleViewer method*), 70
visit() (*robot.model.body.BaseBody method*), 230
visit() (*robot.model.body.Body method*), 232
visit() (*robot.model.body.Branches method*), 234
visit() (*robot.model.control.Break method*), 250
visit() (*robot.model.control.Continue method*), 249
visit() (*robot.model.control.For method*), 239
visit() (*robot.model.control.If method*), 243
visit() (*robot.model.control.IfBranch method*), 241
visit() (*robot.model.control.Return method*), 247
visit() (*robot.model.control.Try method*), 245
visit() (*robot.model.control.TryBranch method*), 244
visit() (*robot.model.control.While method*), 240
visit() (*robot.model.itemlist.ItemList method*), 259
visit() (*robot.model.keyword.Keyword method*), 260
visit() (*robot.model.keyword.Keywords method*), 262
visit() (*robot.model.message.Message method*), 263
visit() (*robot.model.message.Messages method*), 264
visit() (*robot.model.statistics.Statistics method*), 270
visit() (*robot.model.stats.CombinedTagStat method*), 276
visit() (*robot.model.stats.Stat method*), 275
visit() (*robot.model.stats.SuiteStat method*), 275
visit() (*robot.model.stats.TagStat method*), 276
visit() (*robot.model.stats.TotalStat method*), 275
visit() (*robot.model.suitestatistics.SuiteStatistics method*), 276
visit() (*robot.model.tagstatistics.TagStatistics method*), 282
visit() (*robot.model.testcase.TestCase method*), 284
visit() (*robot.model.testcase.TestCases method*), 285
visit() (*robot.model.testsuite.TestSuite method*), 287
visit() (*robot.model.testsuite.TestSuites method*), 288
visit() (*robot.model.totalstatistics.TotalStatistics method*), 288
visit() (*robot.output.loggerhelper.Message method*), 311
visit() (*robot.parsing.model.blocks.FirstStatementFinder method*), 345
visit() (*robot.parsing.model.blocks.LastStatementFinder method*), 345
visit() (*robot.parsing.model.blocks.ModelValidator method*), 345
visit() (*robot.parsing.model.blocks.ModelWriter method*), 345
visit() (*robot.parsing.model.visitor.ModelTransformer method*), 389
visit() (*robot.parsing.model.visitor.ModelVisitor method*), 389
visit() (*robot.parsing.parser.parser.SetLanguages method*), 392
visit() (*robot.parsing.suitestructure.SuiteStructure method*), 392
visit() (*robot.result.executionerrors.ExecutionErrors method*), 413
visit() (*robot.result.executionresult.CombinedResult method*), 415
visit() (*robot.result.executionresult.Result method*), 414
visit() (*robot.result.model.Body method*), 459
visit() (*robot.result.model.Branches method*), 461
visit() (*robot.result.model.Break method*), 490
visit() (*robot.result.model.Continue method*), 487
visit() (*robot.result.model.For method*), 469
visit() (*robot.result.model.ForIteration method*), 465
visit() (*robot.result.model.If method*), 478
visit() (*robot.result.model.IfBranch method*), 476
visit() (*robot.result.model.Iterations method*), 463
visit() (*robot.result.model.Keyword method*), 493
visit() (*robot.result.model.Message method*), 464
visit() (*robot.result.model.Return method*), 485
visit() (*robot.result.model.TestCase method*), 495
visit() (*robot.result.model.TestSuite method*), 498
visit() (*robot.result.model.Try method*), 483
visit() (*robot.result.model.TryBranch method*), 480
visit() (*robot.result.model.While method*), 474
visit() (*robot.result.model.WhileIteration method*), 469
visit() (*robot.running.builder.parsers.ErrorReporter method*), 543
visit() (*robot.running.builder.transformers.ForBuilder method*), 546
visit() (*robot.running.builder.transformers.IfBuilder method*), 547
visit() (*robot.running.builder.transformers.KeywordBuilder method*), 546
visit() (*robot.running.builder.transformers.ResourceBuilder method*), 545
visit() (*robot.running.builder.transformers.SettingsBuilder*

method), 544
visit() (*robot.running.builder.transformers.SuiteBuilder method*), 544
visit() (*robot.running.builder.transformers.TestCaseBuilder method*), 545
visit() (*robot.running.builder.transformers.TryBuilder method*), 547
visit() (*robot.running.builder.transformers.WhileBuilder method*), 548
visit() (*robot.running.model.Body method*), 555
visit() (*robot.running.model.Break method*), 570
visit() (*robot.running.model.Continue method*), 568
visit() (*robot.running.model.For method*), 559
visit() (*robot.running.model.If method*), 563
visit() (*robot.running.model.IfBranch method*), 562
visit() (*robot.running.model.Imports method*), 577
visit() (*robot.running.model.Keyword method*), 557
visit() (*robot.running.model.Return method*), 567
visit() (*robot.running.model.TestCase method*), 572
visit() (*robot.running.model.TestSuite method*), 576
visit() (*robot.running.model.Try method*), 566
visit() (*robot.running.model.TryBranch method*), 564
visit() (*robot.running.model.While method*), 560
visit_Arguments() (*robot.running.builder.transformers.KeywordBuilder method*), 546
visit_Block() (*robot.parsing.model.blocks.ModelValidator method*), 345
visit_break() (*robot.conf.gatherfailed.GatherFailedSuites method*), 30
visit_break() (*robot.conf.gatherfailed.GatherFailedTests method*), 26
visit_break() (*robot.model.configurer.SuiteConfigurer method*), 237
visit_break() (*robot.model.filter.EmptySuiteRemover method*), 253
visit_break() (*robot.model.filter.Filter method*), 257
visit_break() (*robot.model.modifier.ModelModifier method*), 268
visit_break() (*robot.model.statistics.StatisticsBuilder method*), 273
visit_break() (*robot.model.tagsetter.TagSetter method*), 280
visit_break() (*robot.model.totalstatistics.TotalStatisticsBuilder method*), 291
visit_break() (*robot.model.visitor.SuiteVisitor method*), 297
visit_break() (*robot.output.console.dotted.StatusReporter method*), 302
visit_break() (*robot.output.xmllogger.XmlLogger method*), 316
visit_break() (*robot.reporting.outputwriter.OutputWriter method*), 399
visit_break() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 406
visit_break() (*robot.result.configurer.SuiteConfigurer method*), 411
visit_break() (*robot.result.keywordremover.AllKeywordsRemover method*), 418
visit_break() (*robot.result.keywordremover.ByNameKeywordRemover method*), 426
visit_break() (*robot.result.keywordremover.ByTagKeywordRemover method*), 431
visit_break() (*robot.result.keywordremover.ForLoopItemsRemover method*), 435
visit_break() (*robot.result.keywordremover.PassedKeywordRemover method*), 422
visit_break() (*robot.result.keywordremover.WaitUntilKeywordSucceed method*), 443
visit_break() (*robot.result.keywordremover.WarningAndErrorFinder method*), 448
visit_break() (*robot.result.keywordremover.WhileLoopItemsRemover method*), 439
visit_break() (*robot.result.merger.Merger method*), 452
visit_break() (*robot.result.messagefilter.MessageFilter method*), 456
visit_break() (*robot.result.resultbuilder.RemoveKeywords method*), 504
visit_break() (*robot.result.suiteteardownfailed.SuiteTeardownFailed method*), 512
visit_break() (*robot.result.suiteteardownfailed.SuiteTeardownFailure method*), 508
visit_break() (*robot.result.visitor.ResultVisitor method*), 517
visit_Break() (*robot.running.builder.transformers.ForBuilder method*), 546
visit_Break() (*robot.running.builder.transformers.IfBuilder method*), 547
visit_Break() (*robot.running.builder.transformers.KeywordBuilder method*), 546
visit_Break() (*robot.running.builder.transformers.TestCaseBuilder method*), 545
visit_Break() (*robot.running.builder.transformers.TryBuilder method*), 547
visit_Break() (*robot.running.builder.transformers.WhileBuilder method*), 547
visit_break() (*robot.running.randomizer.Randomizer method*), 581
visit_break() (*robot.running.suiterunner.SuiteRunner method*), 587
visit_Config() (*robot.parsing.parser.parser.SetLanguages method*), 392
visit_continue() (*robot.conf.gatherfailed.GatherFailedSuites method*), 30
visit_continue() (*robot.conf.gatherfailed.GatherFailedTests method*), 26

visit_continue() (robot.model.configurer.SuiteConfigurer method), 237
visit_continue() (robot.model.filter.EmptySuiteRemover method), 253
visit_continue() (robot.model.filter.Filter method), 257
visit_continue() (robot.model.modifier.ModelModifier method), 268
visit_continue() (robot.model.statistics.StatisticsBuilder method), 273
visit_continue() (robot.model.tagsetter.TagSetter method), 280
visit_continue() (robot.model.totalstatistics.TotalStatisticsBuilder method), 291
visit_continue() (robot.model.visitor.SuiteVisitor method), 297
visit_continue() (robot.output.console.dotted.StatusReporter method), 302
visit_continue() (robot.output.xmllogger.XmlLogger method), 316
visit_continue() (robot.reporting.outputwriter.OutputWriter method), 399
visit_continue() (robot.reporting.xunitwriter.XUnitFileWriter method), 406
visit_continue() (robot.result.configurer.SuiteConfigurer method), 411
visit_continue() (robot.result.keywordremover.AllKeywordsRemover method), 418
visit_continue() (robot.result.keywordremover.ByNameKeywordRemover method), 426
visit_continue() (robot.result.keywordremover.ByTagKeywordRemover method), 431
visit_continue() (robot.result.keywordremover.ForLoopItemsRemover method), 435
visit_continue() (robot.result.keywordremover.PassedKeywordRemover method), 423
visit_continue() (robot.result.keywordremover.WaitUntilKeywordSucceedsRemover method), 443
visit_continue() (robot.result.keywordremover.WarningAndErrorFinder method), 448
visit_continue() (robot.result.keywordremover.WhileLoopItemsRemover method), 439
visit_continue() (robot.result.merger.Merger method), 452
visit_continue() (robot.result.messagefilter.MessageFilter method), 456
visit_continue() (robot.result.resultbuilder.RemoveKeywords method), 504
visit_continue() (robot.result.suiteteardownfailed.SuiteTeardownFailed method), 512
visit_continue() (robot.result.suiteteardownfailed.SuiteTeardownFailureHandler method), 508
visit_continue() (robot.result.visitor.ResultVisitor method), 517
visit_Continue() (robot.running.builder.transformers.ForBuilder method), 546
visit_Continue() (robot.running.builder.transformers.IfBuilder method), 547
visit_Continue() (robot.running.builder.transformers.KeywordBuilder method), 546
visit_Continue() (robot.running.builder.transformers.TestCaseBuilder method), 545
visit_Continue() (robot.running.builder.transformers.TryBuilder method), 547
visit_Continue() (robot.running.builder.transformers.WhileBuilder method), 548
visit_Continue() (robot.running.randomizer.Randomizer method), 581
visit_continue() (robot.running.suiterunner.SuiteRunner method), 587
visit_DefaultTags() (robot.running.builder.transformers.SettingsBuilder method), 544
visit_directory() (robot.parsing.suitestructure.SuiteStructureVisitor method), 393
visit_directory() (robot.running.builder.builders.SuiteStructureParser method), 542
visit_Documentation() (robot.running.builder.transformers.KeywordBuilder method), 545
visit_Documentation() (robot.running.builder.transformers.ResourceBuilder method), 545
visit_Documentation() (robot.running.builder.transformers.SettingsBuilder method), 544
visit_Documentation() (robot.running.builder.transformers.TestCaseBuilder method), 545
visit_Error() (robot.running.builder.parsers.ErrorReporter method), 543
visit_errors() (robot.output.xmllogger.XmlLogger method), 317
visit_errors() (robot.reporting.outputwriter.OutputWriter method), 399
visit_errors() (robot.reporting.xunitwriter.XUnitFileWriter method), 402
visit_errors() (robot.result.visitor.ResultVisitor method), 514
visit_file() (robot.parsing.suitestructure.SuiteStructureVisitor method), 393
visit_file() (robot.running.builder.builders.SuiteStructureParser method), 542
visit_for() (robot.conf.gatherfailed.GatherFailedSuites method), 30
visit_for() (robot.conf.gatherfailed.GatherFailedTests

method), 26
visit_for() (*robot.model.configurer.SuiteConfigurer method*), 237
visit_for() (*robot.model.filter.EmptySuiteRemover method*), 253
visit_for() (*robot.model.filter.Filter method*), 257
visit_for() (*robot.model.modifier.ModelModifier method*), 268
visit_for() (*robot.model.statistics.StatisticsBuilder method*), 273
visit_for() (*robot.model.tagsetter.TagSetter method*), 280
visit_for() (*robot.model.totalstatistics.TotalStatisticsBuilder method*), 292
visit_for() (*robot.model.visitor.SuiteVisitor method*), 295
visit_for() (*robot.output.console.dotted.StatusReporter method*), 302
visit_for() (*robot.output.xmllogger.XmlLogger method*), 317
visit_for() (*robot.reporting.outputwriter.OutputWriter method*), 399
visit_for() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 406
visit_for() (*robot.result.configurer.SuiteConfigurer method*), 411
visit_for() (*robot.result.keywordremover.AllKeywordsRemover method*), 415
visit_for() (*robot.result.keywordremover.ByNameKeywordRemover method*), 427
visit_for() (*robot.result.keywordremover.ByTagKeywordRemover method*), 431
visit_for() (*robot.result.keywordremover.ForLoopItemsRemover method*), 435
visit_for() (*robot.result.keywordremover.PassedKeywordRemover method*), 423
visit_for() (*robot.result.keywordremover.WaitUntilKeywordSucceedsRemover method*), 443
visit_for() (*robot.result.keywordremover.WarningAndErrorFinder method*), 448
visit_for() (*robot.result.keywordremover.WhileLoopItemsRemover method*), 439
visit_for() (*robot.result.merger.Merger method*), 452
visit_for() (*robot.result.messagefilter.MessageFilter method*), 456
visit_for() (*robot.result.resultbuilder.RemoveKeywords method*), 504
visit_for() (*robot.result.suiteteardownfailed.SuiteTeardownFailed method*), 512
visit_for() (*robot.result.suiteteardownfailed.SuiteTeardownFailureHandler method*), 508
visit_for() (*robot.result.visitor.ResultVisitor method*), 517
visit_For() (*robot.running.builder.transformers.ForBuilder method*), 546
visit_For() (*robot.running.builder.transformers.IfBuilder method*), 547
visit_For() (*robot.running.builder.transformers.KeywordBuilder method*), 546
visit_For() (*robot.running.builder.transformers.TestCaseBuilder method*), 545
visit_For() (*robot.running.builder.transformers.TryBuilder method*), 547
visit_For() (*robot.running.builder.transformers.WhileBuilder method*), 547
visit_for() (*robot.running.randomizer.Randomizer method*), 581
visit_for() (*robot.running.suiterunner.SuiteRunner method*), 588
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*), 237
visit_for_iteration() (*robot.model.filter.EmptySuiteRemover method*), 253
visit_for_iteration() (*robot.model.filter.Filter method*), 257
visit_for_iteration() (*robot.model.modifier.ModelModifier method*), 268
visit_for_iteration() (*robot.model.statistics.StatisticsBuilder method*), 273
visit_for_iteration() (*robot.model.tagsetter.TagSetter method*), 281
visit_for_iteration() (*robot.model.totalstatistics.TotalStatisticsBuilder method*), 292
visit_for_iteration() (*robot.model.visitor.SuiteVisitor method*), 295
visit_for_iteration() (*robot.output.console.dotted.StatusReporter method*), 302
visit_for_iteration() (*robot.output.xmllogger.XmlLogger method*), 317
visit_for_iteration() (*robot.reporting.outputwriter.OutputWriter method*), 399

visit_for_iteration() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 406
visit_for_iteration() (*robot.result.configurer.SuiteConfigurer method*), 411
visit_for_iteration() (*robot.result.keywordremover.AllKeywordsRemover method*), 418
visit_for_iteration() (*robot.result.keywordremover.ByNameKeywordRemover method*), 427
visit_for_iteration() (*robot.result.keywordremover.ByTagKeywordRemover method*), 431
visit_for_iteration() (*robot.result.keywordremover.ForLoopItemsRemover method*), 435
visit_for_iteration() (*robot.result.keywordremover.PassedKeywordRemover method*), 423
visit_for_iteration() (*robot.result.keywordremover.WaitUntilKeywordSucceedsRemover method*), 443
visit_for_iteration() (*robot.result.keywordremover.WarningAndErrorFinder method*), 448
visit_for_iteration() (*robot.result.keywordremover.WhileLoopItemsRemover method*), 439
visit_for_iteration() (*robot.result.merger.Merger method*), 452
visit_for_iteration() (*robot.result.messagefilter.MessageFilter method*), 456
visit_for_iteration() (*robot.result.resultbuilder.RemoveKeywords method*), 504
visit_for_iteration() (*robot.result.suiteteardownfailed.SuiteTeardownFailed method*), 512
visit_for_iteration() (*robot.result.suiteteardownfailed.SuiteTeardownFailureHandler method*), 508
visit_for_iteration() (*robot.result.visitor.ResultVisitor method*), 517
visit_for_iteration() (*robot.running.randomizer.Randomizer method*), 581
visit_for_iteration() (*robot.running.suiterunner.SuiteRunner method*), 588
visit_ForceTags() (*robot.running.builder.transformers.SettingsBuilder method*), 544
visit_if() (*robot.conf.gatherfailed.GatherFailedSuites method*), 31
visit_if() (*robot.conf.gatherfailed.GatherFailedTests method*), 26
visit_if() (*robot.model.configurer.SuiteConfigurer method*), 237
visit_if() (*robot.model.filter.EmptySuiteRemover method*), 253
visit_if() (*robot.model.filter.Filter method*), 257
visit_if() (*robot.model.modifier.ModelModifier method*), 269
visit_if() (*robot.model.statistics.StatisticsBuilder method*), 273
visit_if() (*robot.model.tagsetter.TagSetter method*), 281
visit_if() (*robot.model.totalstatistics.TotalStatisticsBuilder method*), 292
visit_if() (*robot.model.visitor.SuiteVisitor method*), 295
visit_if() (*robot.output.console.dotted.StatusReporter method*), 302
visit_if() (*robot.output.xmllogger.XmlLogger method*), 317
visit_if() (*robot.reporting.outputwriter.OutputWriter method*), 399
visit_if() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 406
visit_if() (*robot.result.configurer.SuiteConfigurer method*), 411
visit_if() (*robot.result.keywordremover.AllKeywordsRemover method*), 419
visit_if() (*robot.result.keywordremover.ByNameKeywordRemover method*), 427
visit_if() (*robot.result.keywordremover.ByTagKeywordRemover method*), 431
visit_if() (*robot.result.keywordremover.ForLoopItemsRemover method*), 435
visit_if() (*robot.result.keywordremover.PassedKeywordRemover method*), 423
visit_if() (*robot.result.keywordremover.WaitUntilKeywordSucceedsRemover method*), 444
visit_if() (*robot.result.keywordremover.WarningAndErrorFinder method*), 448
visit_if() (*robot.result.keywordremover.WhileLoopItemsRemover method*), 439
visit_if() (*robot.result.merger.Merger method*), 452
visit_if() (*robot.result.messagefilter.MessageFilter method*), 456
visit_if() (*robot.result.resultbuilder.RemoveKeywords method*), 504
visit_if() (*robot.result.suiteteardownfailed.SuiteTeardownFailed method*), 512

visit_if() (*robot.result.suiteteardownfailed.SuiteTeardownFailureHandler method*), 508
visit_if() (*robot.result.visitor.ResultVisitor method*), 517
visit_If() (*robot.running.builder.transformers.ForBuilder method*), 546
visit_If() (*robot.running.builder.transformers.IfBuilder method*), 547
visit_If() (*robot.running.builder.transformers.KeywordBuilder method*), 546
visit_If() (*robot.running.builder.transformers.TestCaseBuilder method*), 545
visit_If() (*robot.running.builder.transformers.TryBuilder method*), 547
visit_If() (*robot.running.builder.transformers.WhileBuilder method*), 547
visit_if() (*robot.running.randomizer.Randomizer method*), 582
visit_if() (*robot.running.suiterunner.SuiteRunner method*), 588
visit_if_branch() (*robot.conf.gatherfailed.GatherFailedSuites method*), 31
visit_if_branch() (*robot.conf.gatherfailed.GatherFailedTests method*), 26
visit_if_branch() (*robot.model.configurer.SuiteConfigurer method*), 237
visit_if_branch() (*robot.model.filter.EmptySuiteRemover method*), 253
visit_if_branch() (*robot.model.filter.Filter method*), 258
visit_if_branch() (*robot.model.modifier.ModelModifier method*), 269
visit_if_branch() (*robot.model.statistics.StatisticsBuilder method*), 274
visit_if_branch() (*robot.model.tagsetter.TagSetter method*), 281
visit_if_branch() (*robot.model.totalstatistics.TotalStatisticsBuilder method*), 292
visit_if_branch() (*robot.model.visitor.SuiteVisitor method*), 296
visit_if_branch() (*robot.output.console.dotted.StatusReporter method*), 302
visit_if_branch() (*robot.output.xmllogger.XmlLogger method*), 317
visit_if_branch() (*robot.reporting.outputwriter.OutputWriter method*), 399
visit_if_branch() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 406
visit_if_branch() (*robot.result.configurer.SuiteConfigurer method*), 412
visit_if_branch() (*robot.result.keywordremover.AllKeywordsRemover method*), 415
visit_if_branch() (*robot.result.keywordremover.ByNameKeywordRemover method*), 427
visit_if_branch() (*robot.result.keywordremover.ByTagKeywordRemover method*), 431
visit_if_branch() (*robot.result.keywordremover.ForLoopItemsRemover method*), 435
visit_if_branch() (*robot.result.keywordremover.PassedKeywordRemover method*), 423
visit_if_branch() (*robot.result.keywordremover.WaitUntilKeywordSucceedsRemover method*), 444
visit_if_branch() (*robot.result.keywordremover.WarningAndErrorFinder method*), 448
visit_if_branch() (*robot.result.keywordremover.WhileLoopItemsRemover method*), 439
visit_if_branch() (*robot.result.merger.Merger method*), 452
visit_if_branch() (*robot.result.messagefilter.MessageFilter method*), 456
visit_if_branch() (*robot.result.resultbuilder.RemoveKeywords method*), 504
visit_if_branch() (*robot.result.suiteteardownfailed.SuiteTeardownFailed method*), 513
visit_if_branch() (*robot.result.suiteteardownfailed.SuiteTeardownFailureHandler method*), 508
visit_if_branch() (*robot.result.visitor.ResultVisitor method*), 518
visit_if_branch() (*robot.running.randomizer.Randomizer method*), 582

visit_if_branch() (robot.running.suiterunner.SuiteRunner method), 588
visit_keyword() (robot.conf.gatherfailed.GatherFailedSuites method), 27
visit_keyword() (robot.conf.gatherfailed.GatherFailedTests method), 23
visit_keyword() (robot.model.configurer.SuiteConfigurer method), 238
visit_keyword() (robot.model.filter.EmptySuiteRemover method), 250
visit_keyword() (robot.model.filter.Filter method), 258
visit_keyword() (robot.model.modifier.ModelModifier method), 269
visit_keyword() (robot.model.statistics.StatisticsBuilder method), 270
visit_keyword() (robot.model.tagsetter.TagSetter method), 278
visit_keyword() (robot.model.totalstatistics.TotalStatisticsBuilder method), 289
visit_keyword() (robot.model.visitor.SuiteVisitor method), 294
visit_keyword() (robot.output.console.dotted.StatusReporter method), 302
visit_keyword() (robot.output.xmllogger.XmlLogger method), 317
visit_keyword() (robot.reporting.outputwriter.OutputWriter method), 399
visit_keyword() (robot.reporting.xunitwriter.XUnitFileWriter method), 402
visit_keyword() (robot.result.configurer.SuiteConfigurer method), 412
visit_keyword() (robot.result.keywordremover.AllKeywordsRemover method), 415
visit_keyword() (robot.result.keywordremover.ByNameKeywordRemover method), 427
visit_keyword() (robot.result.keywordremover.ByTagKeywordRemover method), 431
visit_keyword() (robot.result.keywordremover.ForLoopItemsRemover method), 435
visit_keyword() (robot.result.keywordremover.PassedKeywordRemover method), 420
visit_keyword() (robot.result.keywordremover.WaitUntilKeywordSucceedsRemover method), 444
visit_keyword() (robot.result.keywordremover.WarningAndErrorFinder method), 448
visit_keyword() (robot.result.keywordremover.WhileLoopItemsRemover method), 440
visit_keyword() (robot.result.merger.Merger method), 452
visit_keyword() (robot.result.messagefilter.MessageFilter method), 457
visit_keyword() (robot.result.resultbuilder.RemoveKeywords method), 504
visit_keyword() (robot.result.suiteteardownfailed.SuiteTeardownFailed method), 509
visit_keyword() (robot.result.suiteteardownfailed.SuiteTeardownFailureHandler method), 505
visit_keyword() (robot.result.visitor.ResultVisitor method), 518
visit_Keyword() (robot.running.builder.transformers.KeywordBuilder method), 545
visit_Keyword() (robot.running.builder.transformers.ResourceBuilder method), 545
visit_Keyword() (robot.running.builder.transformers.SuiteBuilder method), 544
visit_keyword() (robot.running.randomizer.Randomizer method), 578
visit_keyword() (robot.running.suiterunner.SuiteRunner method), 588
visit_KeywordCall() (robot.running.builder.transformers.ForBuilder method), 546
visit_KeywordCall() (robot.running.builder.transformers.IfBuilder method), 546
visit_KeywordCall() (robot.running.builder.transformers.KeywordBuilder method), 546
visit_KeywordCall() (robot.running.builder.transformers.TestCaseBuilder method), 545
visit_KeywordCall() (robot.running.builder.transformers.TryBuilder method), 547
visit_KeywordCall() (robot.running.builder.transformers.WhileBuilder method), 547
visit_KeywordSection() (robot.running.builder.transformers.SettingsBuilder method), 544
visit_KeywordTags() (robot.running.builder.transformers.ResourceBuilder method), 545
visit_KeywordTags() (robot.running.builder.transformers.SettingsBuilder method), 544
visit_LibraryImport() (robot.running.builder.transformers.ResourceBuilder method), 545
visit_LibraryImport() (robot.running.builder.transformers.SettingsBuilder method), 544
visit_message() (robot.conf.gatherfailed.GatherFailedSuites method), 31
visit_message() (robot.conf.gatherfailed.GatherFailedTests method), 27

visit_message() (*robot.model.configurer.SuiteConfigurer method*), 238
visit_message() (*robot.model.filter.EmptySuiteRemover method*), 254
visit_message() (*robot.model.filter.Filter method*), 258
visit_message() (*robot.model.modifier.ModelModifier method*), 269
visit_message() (*robot.model.statistics.StatisticsBuilder method*), 274
visit_message() (*robot.model.tagsetter.TagSetter method*), 281
visit_message() (*robot.model.totalstatistics.TotalStatisticsBuilder method*), 292
visit_message() (*robot.model.visitor.SuiteVisitor method*), 298
visit_message() (*robot.output.console.dotted.StatusReporter method*), 302
visit_message() (*robot.output.xmllogger.XmlLogger method*), 317
visit_message() (*robot.reporting.outputwriter.OutputWriter method*), 399
visit_message() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 406
visit_message() (*robot.result.configurer.SuiteConfigurer method*), 412
visit_message() (*robot.result.keywordremover.AllKeywordsRemover method*), 419
visit_message() (*robot.result.keywordremover.ByNameKeywordRemover method*), 427
visit_message() (*robot.result.keywordremover.ByTagKeywordRemover method*), 431
visit_message() (*robot.result.keywordremover.ForLoopItemsRemover method*), 435
visit_message() (*robot.result.keywordremover.PassedKeywordRemover method*), 423
visit_message() (*robot.result.keywordremover.WaitUntilKeywordSucceedsRemover method*), 444
visit_message() (*robot.result.keywordremover.WarningAndErrorFinder method*), 445
visit_message() (*robot.result.keywordremover.WhileLoopItemsRemover method*), 440
visit_message() (*robot.result.merger.Merger method*), 453
visit_message() (*robot.result.messagefilter.MessageFilter method*), 457
visit_message() (*robot.result.resultbuilder.RemoveKeywords method*), 504
visit_message() (*robot.result.suiteteardownfailed.SuiteTeardownFailed method*), 513
visit_message() (*robot.result.suiteteardownfailed.SuiteTeardownFailureHandler method*), 509
visit_message() (*robot.result.visitor.ResultVisitor method*), 518
visit_message() (*robot.running.randomizer.Randomizer method*), 582
visit_message() (*robot.running.suiterunner.SuiteRunner method*), 588
visit_Metadata() (*robot.running.builder.transformers.SettingsBuilder method*), 544
visit_ResourceImport() (*robot.running.builder.transformers.ResourceBuilder method*), 545
visit_ResourceImport() (*robot.running.builder.transformers.SettingsBuilder method*), 544
visit_result() (*robot.output.xmllogger.XmlLogger method*), 317
visit_result() (*robot.reporting.outputwriter.OutputWriter method*), 400
visit_result() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 406
visit_result() (*robot.result.visitor.ResultVisitor method*), 514
visit_return() (*robot.conf.gatherfailed.GatherFailedSuites method*), 31
visit_return() (*robot.conf.gatherfailed.GatherFailedTests method*), 27
visit_return() (*robot.model.configurer.SuiteConfigurer method*), 238
visit_return() (*robot.model.filter.EmptySuiteRemover method*), 254
visit_return() (*robot.model.filter.Filter method*), 258
visit_return() (*robot.model.modifier.ModelModifier method*), 269
visit_return() (*robot.model.statistics.StatisticsBuilder method*), 274
visit_return() (*robot.model.tagsetter.TagSetter method*), 281
visit_return() (*robot.model.totalstatistics.TotalStatisticsBuilder method*), 292
visit_return() (*robot.model.visitor.SuiteVisitor method*), 297
visit_return() (*robot.output.console.dotted.StatusReporter method*), 302
visit_return() (*robot.output.xmllogger.XmlLogger method*), 317
visit_return() (*robot.reporting.outputwriter.OutputWriter method*), 400
visit_return() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 406
visit_return() (*robot.result.configurer.SuiteConfigurer method*), 412
visit_return() (*robot.result.keywordremover.AllKeywordsRemover method*), 419
visit_return() (*robot.result.keywordremover.ByNameKeywordRemover method*), 427

visit_return() (robot.result.keywordremover.ByTagKeywordRemover method), 431
visit_return() (robot.result.keywordremover.ForLoopItemsRemover method), 436
visit_return() (robot.result.keywordremover.PassedKeywordRemover method), 423
visit_return() (robot.result.keywordremover.WaitUntilKeywordSucceedsRemover method), 444
visit_return() (robot.result.keywordremover.WarningAndErrorFinder method), 448
visit_return() (robot.result.keywordremover.WhileLoopItemsRemover method), 440
visit_return() (robot.result.merger.Merger method), 453
visit_return() (robot.result.messagefilter.MessageFilter method), 457
visit_return() (robot.result.resultbuilder.RemoveKeywords method), 504
visit_return() (robot.result.suiteteardownfailed.SuiteTeardownFailed method), 513
visit_return() (robot.result.suiteteardownfailed.SuiteTeardownFailureHandler method), 509
visit_return() (robot.result.visitor.ResultVisitor method), 518
visit_Return() (robot.running.builder.transformers.KeywordBuilder method), 546
visit_return() (robot.running.randomizer.Randomizer method), 582
visit_return() (robot.running.suiterunner.SuiteRunner method), 588
visit_ReturnStatement() (robot.running.builder.transformers.ForBuilder method), 546
visit_ReturnStatement() (robot.running.builder.transformers.IfBuilder method), 547
visit_ReturnStatement() (robot.running.builder.transformers.KeywordBuilder method), 546
visit_ReturnStatement() (robot.running.builder.transformers.TestCaseBuilder method), 545
visit_ReturnStatement() (robot.running.builder.transformers.TryBuilder method), 547
visit_ReturnStatement() (robot.running.builder.transformers.WhileBuilder method), 547
visit_SettingSection() (robot.running.builder.transformers.SuiteBuilder method), 544
visit_Setup() (robot.running.builder.transformers.TestCaseBuilder method), 545
visit_stat() (robot.output.xmllogger.XmlLogger method), 316
visit_stat() (robot.reporting.outputwriter.OutputWriter method), 400
visit_stat() (robot.reporting.xunitwriter.XUnitFileWriter method), 406
visit_stat() (robot.result.visitor.ResultVisitor method), 514
visit_Statement() (robot.parsing.model.blocks.FirstStatementFinder method), 345
visit_Statement() (robot.parsing.model.blocks.LastStatementFinder method), 345
visit_Statement() (robot.parsing.model.blocks.ModelValidator method), 345
visit_Statement() (robot.parsing.model.blocks.ModelWriter method), 344
visit_statistics() (robot.output.xmllogger.XmlLogger method), 317
visit_statistics() (robot.reporting.outputwriter.OutputWriter method), 400
visit_statistics() (robot.reporting.xunitwriter.XUnitFileWriter method), 402
visit_statistics() (robot.result.visitor.ResultVisitor method), 514
visit_suite() (robot.conf.gatherfailed.GatherFailedSuites method), 31
visit_suite() (robot.conf.gatherfailed.GatherFailedTests method), 27
visit_suite() (robot.model.configurer.SuiteConfigurer method), 234
visit_suite() (robot.model.filter.EmptySuiteRemover method), 254
visit_suite() (robot.model.filter.Filter method), 258
visit_suite() (robot.model.modifier.ModelModifier method), 265
visit_suite() (robot.model.statistics.StatisticsBuilder method), 274
visit_suite() (robot.model.tagsetter.TagSetter method), 281
visit_suite() (robot.model.totalstatistics.TotalStatisticsBuilder method), 292
visit_suite() (robot.model.visitor.SuiteVisitor method), 294
visit_suite() (robot.output.console.dotted.StatusReporter method), 302
visit_suite() (robot.output.xmllogger.XmlLogger

method), 317
visit_suite() (*robot.reporting.outputwriter.OutputWriter method*), 400
visit_suite() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 406
visit_suite() (*robot.result.configurer.SuiteConfigurer method*), 408
visit_suite() (*robot.result.keywordremover.AllKeywordsRemover method*), 419
visit_suite() (*robot.result.keywordremover.ByNameKeywordRemover method*), 427
visit_suite() (*robot.result.keywordremover.ByTagKeywordRemover method*), 431
visit_suite() (*robot.result.keywordremover.ForLoopItemsRemover method*), 436
visit_suite() (*robot.result.keywordremover.PassedKeywordRemover method*), 423
visit_suite() (*robot.result.keywordremover.WaitUntilKeywordSucceedsRemover method*), 444
visit_suite() (*robot.result.keywordremover.WarningAndErrorFinder method*), 448
visit_suite() (*robot.result.keywordremover.WhileLoopItemsRemover method*), 440
visit_suite() (*robot.result.merger.Merger method*), 453
visit_suite() (*robot.result.messagefilter.MessageFilter method*), 457
visit_suite() (*robot.result.resultbuilder.RemoveKeywords method*), 504
visit_suite() (*robot.result.suiteteardownfailed.SuiteTeardownFailed method*), 513
visit_suite() (*robot.result.suiteteardownfailed.SuiteTeardownFailureHandler method*), 509
visit_suite() (*robot.result.visitor.ResultVisitor method*), 518
visit_suite() (*robot.running.randomizer.Randomizer method*), 582
visit_suite() (*robot.running.suiterunner.SuiteRunner method*), 588
visit_suite_statistics() (*robot.output.xmllogger.XmlLogger method*), 317
visit_suite_statistics() (*robot.reporting.outputwriter.OutputWriter method*), 400
visit_suite_statistics() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 406
visit_suite_statistics() (*robot.result.visitor.ResultVisitor method*), 514
visit_SuiteSetup() (*robot.running.builder.transformers.SettingsBuilder method*), 544
visit_SuiteTeardown() (*robot.running.builder.transformers.SettingsBuilder method*), 544
visit_tag_statistics() (*robot.output.xmllogger.XmlLogger method*), 317
visit_tag_statistics() (*robot.reporting.outputwriter.OutputWriter method*), 400
visit_tag_statistics() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 406
visit_tag_statistics() (*robot.result.visitor.ResultVisitor method*), 514
visit_Tags() (*robot.running.builder.transformers.KeywordBuilder method*), 546
visit_TestCase() (*robot.running.builder.transformers.TestCaseBuilder method*), 545
visit_Teardown() (*robot.running.builder.transformers.KeywordBuilder method*), 546
visit_Teardown() (*robot.running.builder.transformers.TestCaseBuilder method*), 545
visit_Template() (*robot.running.builder.transformers.TestCaseBuilder method*), 545
visit_TemplateArguments() (*robot.running.builder.transformers.ForBuilder method*), 546
visit_TemplateArguments() (*robot.running.builder.transformers.IfBuilder method*), 546
visit_TemplateArguments() (*robot.running.builder.transformers.TestCaseBuilder method*), 545
visit_TemplateArguments() (*robot.running.builder.transformers.TryBuilder method*), 547
visit_TemplateArguments() (*robot.running.builder.transformers.WhileBuilder method*), 547
visit_test() (*robot.conf.gatherfailed.GatherFailedSuites method*), 27
visit_test() (*robot.conf.gatherfailed.GatherFailedTests method*), 23
visit_test() (*robot.model.configurer.SuiteConfigurer method*), 238
visit_test() (*robot.model.filter.EmptySuiteRemover method*), 250
visit_test() (*robot.model.filter.Filter method*), 258
visit_test() (*robot.model.modifier.ModelModifier method*), 269
visit_test() (*robot.model.statistics.StatisticsBuilder method*), 270
visit_test() (*robot.model.tagsetter.TagSetter*

method), 277
visit_test() (*robot.model.totalstatistics.TotalStatisticsBuilder method*), 288
visit_test() (*robot.model.visitor.SuiteVisitor method*), 294
visit_test() (*robot.output.console.dotted.StatusReporter method*), 299
visit_test() (*robot.output.xmllogger.XmlLogger method*), 317
visit_test() (*robot.reporting.outputwriter.OutputWriter method*), 400
visit_test() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 402
visit_test() (*robot.result.configurer.SuiteConfigurer method*), 412
visit_test() (*robot.result.keywordremover.AllKeywordsRemover method*), 419
visit_test() (*robot.result.keywordremover.ByNameKeywordRemover method*), 427
visit_test() (*robot.result.keywordremover.ByTagKeywordRemover method*), 431
visit_test() (*robot.result.keywordremover.ForLoopItemsRemover method*), 436
visit_test() (*robot.result.keywordremover.PassedKeywordRemover method*), 420
visit_test() (*robot.result.keywordremover.WaitUntilKeywordSucceedsRemover method*), 444
visit_test() (*robot.result.keywordremover.WarningAndErrorFinder method*), 448
visit_test() (*robot.result.keywordremover.WhileLoopItemsRemover method*), 440
visit_test() (*robot.result.merger.Merger method*), 449
visit_test() (*robot.result.messagefilter.MessageFilter method*), 457
visit_test() (*robot.result.resultbuilder.RemoveKeywords method*), 501
visit_test() (*robot.result.suiteteardownfailed.SuiteTeardownFailed method*), 509
visit_test() (*robot.result.suiteteardownfailed.SuiteTeardownFailureHandler method*), 505
visit_test() (*robot.result.visitor.ResultVisitor method*), 518
visit_test() (*robot.running.randomizer.Randomizer method*), 578
visit_test() (*robot.running.suiterunner.SuiteRunner method*), 585
visit_TestCase() (*robot.running.builder.transformers.SuiteBuilder method*), 544
visit_TestCase() (*robot.running.builder.transformers.TestCaseBuilder method*), 545
visit_TestCaseSection() (*robot.running.builder.transformers.SettingsBuilder method*), 544
visit_TestSetup() (*robot.running.builder.transformers.SettingsBuilder method*), 544
visit_TestTeardown() (*robot.running.builder.transformers.SettingsBuilder method*), 544
visit_TestTemplate() (*robot.running.builder.transformers.SettingsBuilder method*), 544
visit_TestTimeout() (*robot.running.builder.transformers.SettingsBuilder method*), 544
visit_Timeout() (*robot.running.builder.transformers.KeywordBuilder method*), 546
visit_Timeout() (*robot.running.builder.transformers.TestCaseBuilder method*), 545
visit_total_statistics() (*robot.output.xmllogger.XmlLogger method*), 318
visit_total_statistics() (*robot.reporting.outputwriter.OutputWriter method*), 400
visit_total_statistics() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 406
visit_total_statistics() (*robot.result.visitor.ResultVisitor method*), 714
visit_try() (*robot.conf.gatherfailed.GatherFailedSuites method*), 31
visit_try() (*robot.conf.gatherfailed.GatherFailedTests method*), 27
visit_try() (*robot.model.configurer.SuiteConfigurer method*), 238
visit_try() (*robot.model.filter.EmptySuiteRemover method*), 254
visit_try() (*robot.model.filter.Filter method*), 258
visit_try() (*robot.model.modifier.ModelModifier method*), 269
visit_try() (*robot.model.statistics.StatisticsBuilder method*), 274
visit_try() (*robot.model.tagsetter.TagSetter method*), 281
visit_try() (*robot.model.totalstatistics.TotalStatisticsBuilder method*), 292
visit_try() (*robot.model.visitor.SuiteVisitor method*), 296
visit_try() (*robot.output.console.dotted.StatusReporter method*), 302
visit_try() (*robot.output.xmllogger.XmlLogger method*), 318
visit_Try() (*robot.parsing.model.blocks.ModelValidator method*), 345
visit_try() (*robot.reporting.outputwriter.OutputWriter*

method), 400
visit_try() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 406
visit_try() (*robot.result.configurer.SuiteConfigurer method*), 412
visit_try() (*robot.result.keywordremover.AllKeywordsRemover method*), 419
visit_try() (*robot.result.keywordremover.ByNameKeywordRemover method*), 427
visit_try() (*robot.result.keywordremover.ByTagKeywordRemover method*), 432
visit_try() (*robot.result.keywordremover.ForLoopItemsRemover method*), 436
visit_try() (*robot.result.keywordremover.PassedKeywordRemover method*), 423
visit_try() (*robot.result.keywordremover.WaitUntilKeywordSucceedsRemover method*), 444
visit_try() (*robot.result.keywordremover.WarningAndErrorFinder method*), 448
visit_try() (*robot.result.keywordremover.WhileLoopItemsRemover method*), 440
visit_try() (*robot.result.merger.Merger method*), 453
visit_try() (*robot.result.messagefilter.MessageFilter method*), 457
visit_try() (*robot.result.resultbuilder.RemoveKeywords method*), 504
visit_try() (*robot.result.suiteteardownfailed.SuiteTeardownFailed method*), 513
visit_try() (*robot.result.suiteteardownfailed.SuiteTeardownFailureHandler method*), 509
visit_try() (*robot.result.visitor.ResultVisitor method*), 518
visit_Try() (*robot.running.builder.transformers.ForBuilder method*), 546
visit_Try() (*robot.running.builder.transformers.IfBuilder method*), 547
visit_Try() (*robot.running.builder.transformers.KeywordBuilder method*), 546
visit_Try() (*robot.running.builder.transformers.TestCaseBuilder method*), 545
visit_Try() (*robot.running.builder.transformers.TryBuilder method*), 547
visit_Try() (*robot.running.builder.transformers.WhileBuilder method*), 547
visit_try() (*robot.running.randomizer.Randomizer method*), 582
visit_try() (*robot.running.suiterunner.SuiteRunner method*), 588
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*), 238
visit_try_branch() (*robot.model.filter.EmptySuiteRemover method*), 254
visit_try_branch() (*robot.model.filter.Filter method*), 258
visit_try_branch() (*robot.model.modifier.ModelModifier method*), 269
visit_try_branch() (*robot.model.statistics.StatisticsBuilder method*), 274
visit_try_branch() (*robot.model.tagsetter.TagSetter method*), 281
visit_try_branch() (*robot.model.totalstatistics.TotalStatisticsBuilder method*), 292
visit_try_branch() (*robot.model.visitor.SuiteVisitor method*), 296
visit_try_branch() (*robot.output.console.dotted.StatusReporter method*), 303
visit_try_branch() (*robot.output.xmllogger.XmlLogger method*), 318
visit_try_branch() (*robot.reporting.outputwriter.OutputWriter method*), 400
visit_try_branch() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 407
visit_try_branch() (*robot.result.configurer.SuiteConfigurer method*), 412
visit_try_branch() (*robot.result.keywordremover.AllKeywordsRemover method*), 419
visit_try_branch() (*robot.result.keywordremover.ByNameKeywordRemover method*), 427
visit_try_branch() (*robot.result.keywordremover.ByTagKeywordRemover method*), 432
visit_try_branch() (*robot.result.keywordremover.ForLoopItemsRemover method*), 436
visit_try_branch() (*robot.result.keywordremover.PassedKeywordRemover method*), 423

visit_try_branch() (*robot.result.keywordremover.WaitUntilKeywordSucceedsRemover method*), 444
visit_try_branch() (*robot.result.keywordremover.WarningAndErrorFinder method*), 448
visit_try_branch() (*robot.result.keywordremover.WhileLoopItemsRemover method*), 440
visit_try_branch() (*robot.result.merger.Merger method*), 453
visit_try_branch() (*robot.result.messagefilter.MessageFilter method*), 457
visit_try_branch() (*robot.result.resultbuilder.RemoveKeywords method*), 505
visit_try_branch() (*robot.result.suiteteardownfailed.SuiteTeardownFailed method*), 513
visit_try_branch() (*robot.result.suiteteardownfailed.SuiteTeardownFailureHandler method*), 509
visit_try_branch() (*robot.result.visitor.ResultVisitor method*), 518
visit_try_branch() (*robot.running.randomizer.Randomizer method*), 582
visit_try_branch() (*robot.running.suiterunner.SuiteRunner method*), 588
visit_Variable() (*robot.running.builder.transformers.ResourceBuilder method*), 545
visit_Variable() (*robot.running.builder.transformers.SuiteBuilder method*), 544
visit_VariableSection() (*robot.running.builder.transformers.SettingsBuilder method*), 544
visit_VariablesImport() (*robot.running.builder.transformers.ResourceBuilder method*), 545
visit_VariablesImport() (*robot.running.builder.transformers.SettingsBuilder method*), 544
visit_while() (*robot.conf.gatherfailed.GatherFailedSuites method*), 31
visit_while() (*robot.conf.gatherfailed.GatherFailedTests method*), 27
visit_while() (*robot.model.configurer.SuiteConfigurer method*), 238
visit_while() (*robot.model.filter.EmptySuiteRemover method*), 254
visit_while() (*robot.model.filter.Filter method*), 258
visit_while() (*robot.model.modifier.ModelModifier method*), 269
visit_while() (*robot.model.statistics.StatisticsBuilder method*), 274
visit_while() (*robot.model.tagsetter.TagSetter method*), 281
visit_while() (*robot.model.totalstatistics.TotalStatisticsBuilder method*), 292
visit_while() (*robot.model.visitor.SuiteVisitor method*), 296
visit_while() (*robot.output.console.dotted.StatusReporter method*), 303
visit_while() (*robot.output.xmllogger.XmlLogger method*), 318
visit_while() (*robot.reporting.outputwriter.OutputWriter method*), 400
visit_while() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 407
visit_while() (*robot.result.configurer.SuiteConfigurer method*), 412
visit_while() (*robot.result.keywordremover.AllKeywordsRemover method*), 419
visit_while() (*robot.result.keywordremover.ByNameKeywordRemover method*), 428
visit_while() (*robot.result.keywordremover.ByTagKeywordRemover method*), 432
visit_while() (*robot.result.keywordremover.ForLoopItemsRemover method*), 436
visit_while() (*robot.result.keywordremover.PassedKeywordRemover method*), 423
visit_while() (*robot.result.keywordremover.WaitUntilKeywordSucceedsRemover method*), 444
visit_while() (*robot.result.keywordremover.WarningAndErrorFinder method*), 449
visit_while() (*robot.result.keywordremover.WhileLoopItemsRemover method*), 440
visit_while() (*robot.result.merger.Merger method*), 453
visit_while() (*robot.result.messagefilter.MessageFilter method*), 457
visit_while() (*robot.result.resultbuilder.RemoveKeywords method*), 505
visit_while() (*robot.result.suiteteardownfailed.SuiteTeardownFailed method*), 513
visit_while() (*robot.result.suiteteardownfailed.SuiteTeardownFailureHandler method*), 509
visit_while() (*robot.result.visitor.ResultVisitor method*), 518
visit_While() (*robot.running.builder.transformers.ForBuilder method*), 546
visit_While() (*robot.running.builder.transformers.IfBuilder method*), 547
visit_While() (*robot.running.builder.transformers.KeywordBuilder*

method), 546
visit_While() (*robot.running.builder.transformers.TestCaseBuilder method*), 545
visit_While() (*robot.running.builder.transformers.TryBuilder method*), 547
visit_While() (*robot.running.builder.transformers.WhileBuilder method*), 547
visit_while() (*robot.running.randomizer.Randomizer method*), 582
visit_while() (*robot.running.suiterunner.SuiteRunner method*), 588
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*), 238
visit_while_iteration() (*robot.model.filter.EmptySuiteRemover method*), 254
visit_while_iteration() (*robot.model.filter.Filter method*), 258
visit_while_iteration() (*robot.model.modifier.ModelModifier method*), 269
visit_while_iteration() (*robot.model.statistics.StatisticsBuilder method*), 274
visit_while_iteration() (*robot.model.tagsetter.TagSetter method*), 281
visit_while_iteration() (*robot.model.totalstatistics.TotalStatisticsBuilder method*), 292
visit_while_iteration() (*robot.model.visitor.SuiteVisitor method*), 297
visit_while_iteration() (*robot.output.console.dotted.StatusReporter method*), 303
visit_while_iteration() (*robot.output.xmllogger.XmlLogger method*), 318
visit_while_iteration() (*robot.reporting.outputwriter.OutputWriter method*), 400
visit_while_iteration() (*robot.reporting.xunitwriter.XUnitFileWriter method*), 407
visit_while_iteration() (*robot.result.configurer.SuiteConfigurer method*), 412
visit_while_iteration() (*robot.result.keywordremover.AllKeywordsRemover method*), 419
visit_while_iteration() (*robot.result.keywordremover.ByNameKeywordRemover method*), 428
visit_while_iteration() (*robot.result.keywordremover.ByTagKeywordRemover method*), 432
visit_while_iteration() (*robot.result.keywordremover.ForLoopItemsRemover method*), 436
visit_while_iteration() (*robot.result.keywordremover.PassedKeywordRemover method*), 423
visit_while_iteration() (*robot.result.keywordremover.WaitUntilKeywordSucceedsRemover method*), 444
visit_while_iteration() (*robot.result.keywordremover.WarningAndErrorFinder method*), 449
visit_while_iteration() (*robot.result.keywordremover.WhileLoopItemsRemover method*), 440
visit_while_iteration() (*robot.result.merger.Merger method*), 453
visit_while_iteration() (*robot.result.messagefilter.MessageFilter method*), 457
visit_while_iteration() (*robot.result.resultbuilder.RemoveKeywords method*), 505
visit_while_iteration() (*robot.result.suiteteardownfailed.SuiteTeardownFailed method*), 513
visit_while_iteration() (*robot.result.suiteteardownfailed.SuiteTeardownFailureHandler method*), 509
visit_while_iteration() (*robot.result.visitor.ResultVisitor method*), 518
visit_while_iteration() (*robot.running.randomizer.Randomizer method*), 582
visit_while_iteration() (*robot.running.suiterunner.SuiteRunner method*), 589
VisitorFinder (*class in robot.parsing.model.visitor*), 388

W

wait_for_process() (*robot.libraries.Process.Process method*),

126
wait_until_created() (*robot.libraries.OperatingSystem.OperatingSystem method*), 116
wait_until_keyword_succeeds() (*robot.libraries.BuiltIn.BuiltIn method*), 98
wait_until_removed() (*robot.libraries.OperatingSystem.OperatingSystem method*), 116
wait_variable() (*robot.libraries.dialogs_py.InputDialog method*), 180
wait_variable() (*robot.libraries.dialogs_py.MessageDialog method*), 166
wait_variable() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 208
wait_variable() (*robot.libraries.dialogs_py.PassFailDialog method*), 222
wait_variable() (*robot.libraries.dialogs_py.SelectionDialog method*), 194
wait_visibility() (*robot.libraries.dialogs_py.InputDialog method*), 180
wait_visibility() (*robot.libraries.dialogs_py.MessageDialog method*), 166
wait_visibility() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 208
wait_visibility() (*robot.libraries.dialogs_py.PassFailDialog method*), 222
wait_visibility() (*robot.libraries.dialogs_py.SelectionDialog method*), 194
wait_window() (*robot.libraries.dialogs_py.InputDialog method*), 180
wait_window() (*robot.libraries.dialogs_py.MessageDialog method*), 166
wait_window() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 208
wait_window() (*robot.libraries.dialogs_py.PassFailDialog method*), 222
wait_window() (*robot.libraries.dialogs_py.SelectionDialog method*), 194
WaitUntilKeywordSucceedsRemover (*class in robot.result.keywordremover*), 440
waitvar() (*robot.libraries.dialogs_py.InputDialog method*), 180
waitvar() (*robot.libraries.dialogs_py.MessageDialog method*), 166
waitvar() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 208
waitvar() (*robot.libraries.dialogs_py.PassFailDialog method*), 222
waitvar() (*robot.libraries.dialogs_py.SelectionDialog method*), 194
warn() (*in module robot.api.logger*), 15
warn() (*in module robot.output.librarylogger*), 305
warn() (*robot.output.filelogger.FileLogger method*), 305
warn() (*robot.output.logger.Logger method*), 309
warn() (*robot.output.loggerhelper.AbstractLogger method*), 309
warn() (*robot.output.output.Output method*), 311
warn() (*robot.utils.importer.NoLogger method*), 602
WarningAndErrorFinder (*class in robot.result.keywordremover*), 445
when_prefixes (*robot.conf.languages.Bg attribute*), 60
when_prefixes (*robot.conf.languages.Bs attribute*), 39
when_prefixes (*robot.conf.languages.Cs attribute*), 36
when_prefixes (*robot.conf.languages.De attribute*), 43
when_prefixes (*robot.conf.languages.En attribute*), 35
when_prefixes (*robot.conf.languages.Es attribute*), 51
when_prefixes (*robot.conf.languages.Fi attribute*), 40
when_prefixes (*robot.conf.languages.Fr attribute*), 42
when_prefixes (*robot.conf.languages.Hi attribute*), 64
when_prefixes (*robot.conf.languages.It attribute*), 63
when_prefixes (*robot.conf.languages.Language attribute*), 33
when_prefixes (*robot.conf.languages.Nl attribute*), 38
when_prefixes (*robot.conf.languages.Pl attribute*), 49
when_prefixes (*robot.conf.languages.Pt attribute*), 46
when_prefixes (*robot.conf.languages.PtBr attribute*), 44
when_prefixes (*robot.conf.languages.Ro attribute*), 61
when_prefixes (*robot.conf.languages.Ru attribute*), 53
when_prefixes (*robot.conf.languages.Sv attribute*), 58
when_prefixes (*robot.conf.languages.Th attribute*), 47
when_prefixes (*robot.conf.languages.Tr attribute*), 57

when_prefixes (*robot.conf.languages.Uk attribute*), 50
when_prefixes (*robot.conf.languages.ZhCn attribute*), 54
when_prefixes (*robot.conf.languages.ZhTw attribute*), 56
While (*class in robot.model.control*), 240
While (*class in robot.parsing.model.blocks*), 344
While (*class in robot.result.model*), 471
While (*class in robot.running.model*), 559
WHILE (*robot.model.body.BodyItem attribute*), 228
WHILE (*robot.model.control.Break attribute*), 249
WHILE (*robot.model.control.Continue attribute*), 248
WHILE (*robot.model.control.For attribute*), 239
WHILE (*robot.model.control.If attribute*), 243
WHILE (*robot.model.control.IfBranch attribute*), 242
WHILE (*robot.model.control.Return attribute*), 247
WHILE (*robot.model.control.Try attribute*), 246
WHILE (*robot.model.control.TryBranch attribute*), 244
WHILE (*robot.model.control.While attribute*), 240
WHILE (*robot.model.keyword.Keyword attribute*), 261
WHILE (*robot.model.message.Message attribute*), 263
WHILE (*robot.output.loggerhelper.Message attribute*), 310
WHILE (*robot.parsing.lexer.tokens.END attribute*), 340
WHILE (*robot.parsing.lexer.tokens.EOS attribute*), 337
WHILE (*robot.parsing.lexer.tokens.Token attribute*), 335
WHILE (*robot.result.model.Break attribute*), 488
WHILE (*robot.result.model.Continue attribute*), 486
WHILE (*robot.result.model.For attribute*), 468
WHILE (*robot.result.model.ForIteration attribute*), 465
WHILE (*robot.result.model.If attribute*), 477
WHILE (*robot.result.model.IfBranch attribute*), 475
WHILE (*robot.result.model.Keyword attribute*), 491
WHILE (*robot.result.model.Message attribute*), 463
WHILE (*robot.result.model.Return attribute*), 484
WHILE (*robot.result.model.Try attribute*), 481
WHILE (*robot.result.model.TryBranch attribute*), 479
WHILE (*robot.result.model.While attribute*), 472
WHILE (*robot.result.model.WhileIteration attribute*), 470
WHILE (*robot.running.model.Break attribute*), 569
WHILE (*robot.running.model.Continue attribute*), 568
WHILE (*robot.running.model.For attribute*), 558
WHILE (*robot.running.model.If attribute*), 562
WHILE (*robot.running.model.IfBranch attribute*), 561
WHILE (*robot.running.model.Keyword attribute*), 556
WHILE (*robot.running.model.Return attribute*), 566
WHILE (*robot.running.model.Try attribute*), 565
WHILE (*robot.running.model.TryBranch attribute*), 564
WHILE (*robot.running.model.While attribute*), 559
while_class (*robot.model.body.BaseBody attribute*), 229
while_class (*robot.model.body.Body attribute*), 232
while_class (*robot.model.body.Branches attribute*), 234
while_class (*robot.result.model.Body attribute*), 459
while_class (*robot.result.model.Branches attribute*), 461
while_class (*robot.result.model.Iterations attribute*), 463
while_class (*robot.running.model.Body attribute*), 555
WhileBuilder (*class in robot.running.builder.transformers*), 547
WhileHandler (*class in robot.result.xmlelementhandlers*), 520
WhileHeader (*class in robot.parsing.model.statements*), 381
WhileHeaderLexer (*class in robot.parsing.lexer.statementlexers*), 332
WhileIteration (*class in robot.result.model*), 469
WhileLexer (*class in robot.parsing.lexer.blocklexers*), 322
WhileLimit (*class in robot.running.bodyrunner*), 550
WhileLoopItemsRemover (*class in robot.result.keywordremover*), 436
WhileParser (*class in robot.parsing.parser.blockparsers*), 390
WhileRunner (*class in robot.running.bodyrunner*), 549
winfo_atom() (*robot.libraries.dialogs_py.InputDialog method*), 181
winfo_atom() (*robot.libraries.dialogs_py.MessageDialog method*), 167
winfo_atom() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 209
winfo_atom() (*robot.libraries.dialogs_py.PassFailDialog method*), 223
winfo_atom() (*robot.libraries.dialogs_py.SelectionDialog method*), 195
winfo_atomname() (*robot.libraries.dialogs_py.InputDialog method*), 181
winfo_atomname() (*robot.libraries.dialogs_py.MessageDialog method*), 167
winfo_atomname() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 209
winfo_atomname() (*robot.libraries.dialogs_py.PassFailDialog method*), 223
winfo_atomname() (*robot.libraries.dialogs_py.SelectionDialog method*), 195
winfo_cells() (*robot.libraries.dialogs_py.InputDialog method*), 181
winfo_cells() (*robot.libraries.dialogs_py.MessageDialog method*), 167
winfo_cells() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 209
winfo_cells() (*robot.libraries.dialogs_py.PassFailDialog*

method), 223
winfo_cells() (*robot.libraries.dialogs_py.SelectionDialog method*), 195
winfo_children() (*robot.libraries.dialogs_py.InputDialog method*), 181
winfo_children() (*robot.libraries.dialogs_py.MessageDialog method*), 167
winfo_children() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 209
winfo_children() (*robot.libraries.dialogs_py.PassFailDialog method*), 223
winfo_children() (*robot.libraries.dialogs_py.SelectionDialog method*), 195
winfo_class() (*robot.libraries.dialogs_py.InputDialog method*), 181
winfo_class() (*robot.libraries.dialogs_py.MessageDialog method*), 167
winfo_class() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 209
winfo_class() (*robot.libraries.dialogs_py.PassFailDialog method*), 223
winfo_class() (*robot.libraries.dialogs_py.SelectionDialog method*), 195
winfo_colormapfull() (*robot.libraries.dialogs_py.InputDialog method*), 181
winfo_colormapfull() (*robot.libraries.dialogs_py.MessageDialog method*), 167
winfo_colormapfull() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 209
winfo_colormapfull() (*robot.libraries.dialogs_py.PassFailDialog method*), 223
winfo_colormapfull() (*robot.libraries.dialogs_py.SelectionDialog method*), 195
winfo_containing() (*robot.libraries.dialogs_py.InputDialog method*), 181
winfo_containing() (*robot.libraries.dialogs_py.MessageDialog method*), 167
winfo_containing() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 209
winfo_containing() (*robot.libraries.dialogs_py.PassFailDialog method*), 223
winfo_containing() (*robot.libraries.dialogs_py.SelectionDialog method*), 195
winfo_depth() (*robot.libraries.dialogs_py.InputDialog method*), 181
winfo_depth() (*robot.libraries.dialogs_py.MessageDialog method*), 167
winfo_depth() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 209
winfo_depth() (*robot.libraries.dialogs_py.PassFailDialog method*), 223
winfo_depth() (*robot.libraries.dialogs_py.SelectionDialog method*), 195
winfo_exists() (*robot.libraries.dialogs_py.InputDialog method*), 181
winfo_exists() (*robot.libraries.dialogs_py.MessageDialog method*), 167
winfo_exists() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 209
winfo_exists() (*robot.libraries.dialogs_py.PassFailDialog method*), 223
winfo_exists() (*robot.libraries.dialogs_py.SelectionDialog method*), 195
winfo_fpixels() (*robot.libraries.dialogs_py.InputDialog method*), 181
winfo_fpixels() (*robot.libraries.dialogs_py.MessageDialog method*), 167
winfo_fpixels() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 209
winfo_fpixels() (*robot.libraries.dialogs_py.PassFailDialog method*), 223
winfo_fpixels() (*robot.libraries.dialogs_py.SelectionDialog method*), 195
winfo_geometry() (*robot.libraries.dialogs_py.InputDialog method*), 181
winfo_geometry() (*robot.libraries.dialogs_py.MessageDialog method*), 167
winfo_geometry() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 209
winfo_geometry() (*robot.libraries.dialogs_py.PassFailDialog method*), 223
winfo_geometry() (*robot.libraries.dialogs_py.SelectionDialog method*), 195
winfo_height() (*robot.libraries.dialogs_py.InputDialog method*), 181
winfo_height() (*robot.libraries.dialogs_py.MessageDialog method*), 167
winfo_height() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 209
winfo_height() (*robot.libraries.dialogs_py.PassFailDialog method*), 223
winfo_height() (*robot.libraries.dialogs_py.SelectionDialog method*), 195
winfo_id() (*robot.libraries.dialogs_py.InputDialog method*), 181
winfo_id() (*robot.libraries.dialogs_py.MessageDialog method*), 167
winfo_id() (*robot.libraries.dialogs_py.MultipleSelectionDialog*

method), 209
winfo_id() (*robot.libraries.dialogs_py.PassFailDialog method*), 223
winfo_id() (*robot.libraries.dialogs_py.SelectionDialog method*), 195
winfo_interps() (*robot.libraries.dialogs_py.InputDialog method*), 181
winfo_interps() (*robot.libraries.dialogs_py.MessageDialog method*), 167
winfo_interps() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 209
winfo_interps() (*robot.libraries.dialogs_py.PassFailDialog method*), 223
winfo_interps() (*robot.libraries.dialogs_py.SelectionDialog method*), 195
winfo_ismapped() (*robot.libraries.dialogs_py.InputDialog method*), 181
winfo_ismapped() (*robot.libraries.dialogs_py.MessageDialog method*), 167
winfo_ismapped() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 209
winfo_ismapped() (*robot.libraries.dialogs_py.PassFailDialog method*), 223
winfo_ismapped() (*robot.libraries.dialogs_py.SelectionDialog method*), 195
winfo_manager() (*robot.libraries.dialogs_py.InputDialog method*), 181
winfo_manager() (*robot.libraries.dialogs_py.MessageDialog method*), 167
winfo_manager() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 209
winfo_manager() (*robot.libraries.dialogs_py.PassFailDialog method*), 223
winfo_manager() (*robot.libraries.dialogs_py.SelectionDialog method*), 195
winfo_name() (*robot.libraries.dialogs_py.InputDialog method*), 181
winfo_name() (*robot.libraries.dialogs_py.MessageDialog method*), 167
winfo_name() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 209
winfo_name() (*robot.libraries.dialogs_py.PassFailDialog method*), 223
winfo_name() (*robot.libraries.dialogs_py.SelectionDialog method*), 195
winfo_parent() (*robot.libraries.dialogs_py.InputDialog method*), 181
winfo_parent() (*robot.libraries.dialogs_py.MessageDialog method*), 167
winfo_parent() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 209
winfo_parent() (*robot.libraries.dialogs_py.PassFailDialog method*), 223
winfo_parent() (*robot.libraries.dialogs_py.SelectionDialog method*), 195
winfo_pathname() (*robot.libraries.dialogs_py.InputDialog method*), 181
winfo_pathname() (*robot.libraries.dialogs_py.MessageDialog method*), 167
winfo_pathname() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 209
winfo_pathname() (*robot.libraries.dialogs_py.PassFailDialog method*), 223
winfo_pathname() (*robot.libraries.dialogs_py.SelectionDialog method*), 195
winfo_pixels() (*robot.libraries.dialogs_py.InputDialog method*), 181
winfo_pixels() (*robot.libraries.dialogs_py.MessageDialog method*), 167
winfo_pixels() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 209
winfo_pixels() (*robot.libraries.dialogs_py.PassFailDialog method*), 223
winfo_pixels() (*robot.libraries.dialogs_py.SelectionDialog method*), 195
winfo_pointerx() (*robot.libraries.dialogs_py.InputDialog method*), 181
winfo_pointerx() (*robot.libraries.dialogs_py.MessageDialog method*), 167
winfo_pointerx() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 209
winfo_pointerx() (*robot.libraries.dialogs_py.PassFailDialog method*), 223
winfo_pointerx() (*robot.libraries.dialogs_py.SelectionDialog method*), 195
winfo_pointerxy() (*robot.libraries.dialogs_py.InputDialog method*), 182
winfo_pointerxy() (*robot.libraries.dialogs_py.MessageDialog method*), 168
winfo_pointerxy() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 210
winfo_pointerxy() (*robot.libraries.dialogs_py.PassFailDialog method*), 224
winfo_pointerxy() (*robot.libraries.dialogs_py.SelectionDialog method*), 196
winfo_pointery() (*robot.libraries.dialogs_py.InputDialog method*), 182
winfo_pointery() (*robot.libraries.dialogs_py.MessageDialog method*), 168
winfo_pointery() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 210
winfo_pointery() (*robot.libraries.dialogs_py.PassFailDialog method*), 224

winfo_pointery() (*robot.libraries.dialogs_py.SelectionDialog method*), 196
winfo_reqheight() (*robot.libraries.dialogs_py.InputDialog method*), 182
winfo_reqheight() (*robot.libraries.dialogs_py.MessageDialog method*), 168
winfo_reqheight() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 210
winfo_reqheight() (*robot.libraries.dialogs_py.PassFailDialog method*), 224
winfo_reqheight() (*robot.libraries.dialogs_py.SelectionDialog method*), 196
winfo_reqwidth() (*robot.libraries.dialogs_py.InputDialog method*), 182
winfo_reqwidth() (*robot.libraries.dialogs_py.MessageDialog method*), 168
winfo_reqwidth() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 210
winfo_reqwidth() (*robot.libraries.dialogs_py.PassFailDialog method*), 224
winfo_reqwidth() (*robot.libraries.dialogs_py.SelectionDialog method*), 196
winfo_rgb() (*robot.libraries.dialogs_py.InputDialog method*), 182
winfo_rgb() (*robot.libraries.dialogs_py.MessageDialog method*), 168
winfo_rgb() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 210
winfo_rgb() (*robot.libraries.dialogs_py.PassFailDialog method*), 224
winfo_rgb() (*robot.libraries.dialogs_py.SelectionDialog method*), 196
winfo_rootx() (*robot.libraries.dialogs_py.InputDialog method*), 182
winfo_rootx() (*robot.libraries.dialogs_py.MessageDialog method*), 168
winfo_rootx() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 210
winfo_rootx() (*robot.libraries.dialogs_py.PassFailDialog method*), 224
winfo_rootx() (*robot.libraries.dialogs_py.SelectionDialog method*), 196
winfo_rooty() (*robot.libraries.dialogs_py.InputDialog method*), 182
winfo_rooty() (*robot.libraries.dialogs_py.MessageDialog method*), 168
winfo_rooty() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 210
winfo_rooty() (*robot.libraries.dialogs_py.PassFailDialog method*), 224
winfo_rooty() (*robot.libraries.dialogs_py.SelectionDialog method*), 196
winfo_screen() (*robot.libraries.dialogs_py.InputDialog method*), 182
winfo_screen() (*robot.libraries.dialogs_py.MessageDialog method*), 168
winfo_screen() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 210
winfo_screen() (*robot.libraries.dialogs_py.PassFailDialog method*), 224
winfo_screen() (*robot.libraries.dialogs_py.SelectionDialog method*), 196
winfo_screencells() (*robot.libraries.dialogs_py.InputDialog method*), 182
winfo_screencells() (*robot.libraries.dialogs_py.MessageDialog method*), 168
winfo_screencells() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 210
winfo_screencells() (*robot.libraries.dialogs_py.PassFailDialog method*), 224
winfo_screencells() (*robot.libraries.dialogs_py.SelectionDialog method*), 196
winfo_screendepth() (*robot.libraries.dialogs_py.InputDialog method*), 182
winfo_screendepth() (*robot.libraries.dialogs_py.MessageDialog method*), 168
winfo_screendepth() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 210
winfo_screendepth() (*robot.libraries.dialogs_py.PassFailDialog method*), 224
winfo_screendepth() (*robot.libraries.dialogs_py.SelectionDialog method*), 196
winfo_screenheight() (*robot.libraries.dialogs_py.InputDialog method*), 182
winfo_screenheight() (*robot.libraries.dialogs_py.MessageDialog method*), 168
winfo_screenheight() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 210
winfo_screenheight() (*robot.libraries.dialogs_py.PassFailDialog*

method), 224
winfo_screenheight() (*robot.libraries.dialogs_py.SelectionDialog method*), 196
winfo_screenmmheight() (*robot.libraries.dialogs_py.InputDialog method*), 182
winfo_screenmmheight() (*robot.libraries.dialogs_py.MessageDialog method*), 168
winfo_screenmmheight() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 210
winfo_screenmmheight() (*robot.libraries.dialogs_py.PassFailDialog method*), 224
winfo_screenmmheight() (*robot.libraries.dialogs_py.SelectionDialog method*), 196
winfo_screenmmwidth() (*robot.libraries.dialogs_py.InputDialog method*), 182
winfo_screenmmwidth() (*robot.libraries.dialogs_py.MessageDialog method*), 168
winfo_screenmmwidth() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 210
winfo_screenmmwidth() (*robot.libraries.dialogs_py.PassFailDialog method*), 224
winfo_screenmmwidth() (*robot.libraries.dialogs_py.SelectionDialog method*), 196
winfo_screenvisual() (*robot.libraries.dialogs_py.InputDialog method*), 182
winfo_screenvisual() (*robot.libraries.dialogs_py.MessageDialog method*), 168
winfo_screenvisual() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 210
winfo_screenvisual() (*robot.libraries.dialogs_py.PassFailDialog method*), 224
winfo_screenvisual() (*robot.libraries.dialogs_py.SelectionDialog method*), 196
winfo_screenwidth() (*robot.libraries.dialogs_py.InputDialog method*), 182
winfo_screenwidth() (*robot.libraries.dialogs_py.MessageDialog method*), 168
winfo_screenwidth() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 210
winfo_screenwidth() (*robot.libraries.dialogs_py.PassFailDialog method*), 224
winfo_screenwidth() (*robot.libraries.dialogs_py.SelectionDialog method*), 196
winfo_server() (*robot.libraries.dialogs_py.InputDialog method*), 182
winfo_server() (*robot.libraries.dialogs_py.MessageDialog method*), 168
winfo_server() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 210
winfo_server() (*robot.libraries.dialogs_py.PassFailDialog method*), 224
winfo_server() (*robot.libraries.dialogs_py.SelectionDialog method*), 196
winfo_toplevel() (*robot.libraries.dialogs_py.InputDialog method*), 182
winfo_toplevel() (*robot.libraries.dialogs_py.MessageDialog method*), 168
winfo_toplevel() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 210
winfo_toplevel() (*robot.libraries.dialogs_py.PassFailDialog method*), 224
winfo_toplevel() (*robot.libraries.dialogs_py.SelectionDialog method*), 196
winfo_viewable() (*robot.libraries.dialogs_py.InputDialog method*), 182
winfo_viewable() (*robot.libraries.dialogs_py.MessageDialog method*), 168
winfo_viewable() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 210
winfo_viewable() (*robot.libraries.dialogs_py.PassFailDialog method*), 224
winfo_viewable() (*robot.libraries.dialogs_py.SelectionDialog method*), 196
winfo_visual() (*robot.libraries.dialogs_py.InputDialog method*), 182
winfo_visual() (*robot.libraries.dialogs_py.MessageDialog method*), 168
winfo_visual() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 210
winfo_visual() (*robot.libraries.dialogs_py.PassFailDialog method*), 224
winfo_visual() (*robot.libraries.dialogs_py.SelectionDialog method*), 196
winfo_visualid() (*robot.libraries.dialogs_py.InputDialog method*), 182
winfo_visualid() (*robot.libraries.dialogs_py.MessageDialog method*), 168

winfo_visualid() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 210
winfo_visualid() (*robot.libraries.dialogs_py.PassFailDialog method*), 224
winfo_visualid() (*robot.libraries.dialogs_py.SelectionDialog method*), 196
winfo_visualsavailable() (*robot.libraries.dialogs_py.InputDialog method*), 183
winfo_visualsavailable() (*robot.libraries.dialogs_py.MessageDialog method*), 169
winfo_visualsavailable() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 211
winfo_visualsavailable() (*robot.libraries.dialogs_py.PassFailDialog method*), 225
winfo_visualsavailable() (*robot.libraries.dialogs_py.SelectionDialog method*), 197
winfo_vrootheight() (*robot.libraries.dialogs_py.InputDialog method*), 183
winfo_vrootheight() (*robot.libraries.dialogs_py.MessageDialog method*), 169
winfo_vrootheight() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 211
winfo_vrootheight() (*robot.libraries.dialogs_py.PassFailDialog method*), 225
winfo_vrootheight() (*robot.libraries.dialogs_py.SelectionDialog method*), 197
winfo_vrootwidth() (*robot.libraries.dialogs_py.InputDialog method*), 183
winfo_vrootwidth() (*robot.libraries.dialogs_py.MessageDialog method*), 169
winfo_vrootwidth() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 211
winfo_vrootwidth() (*robot.libraries.dialogs_py.PassFailDialog method*), 225
winfo_vrootwidth() (*robot.libraries.dialogs_py.SelectionDialog method*), 197
winfo_vrootx() (*robot.libraries.dialogs_py.InputDialog method*), 183
winfo_vrootx() (*robot.libraries.dialogs_py.MessageDialog method*), 169
winfo_vrootx() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 211
winfo_vrootx() (*robot.libraries.dialogs_py.PassFailDialog method*), 225
winfo_vrootx() (*robot.libraries.dialogs_py.SelectionDialog method*), 197
winfo_vrooty() (*robot.libraries.dialogs_py.InputDialog method*), 183
winfo_vrooty() (*robot.libraries.dialogs_py.MessageDialog method*), 169
winfo_vrooty() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 211
winfo_vrooty() (*robot.libraries.dialogs_py.PassFailDialog method*), 225
winfo_vrooty() (*robot.libraries.dialogs_py.SelectionDialog method*), 197
winfo_width() (*robot.libraries.dialogs_py.InputDialog method*), 183
winfo_width() (*robot.libraries.dialogs_py.MessageDialog method*), 169
winfo_width() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 211
winfo_width() (*robot.libraries.dialogs_py.PassFailDialog method*), 225
winfo_width() (*robot.libraries.dialogs_py.SelectionDialog method*), 197
winfo_x() (*robot.libraries.dialogs_py.InputDialog method*), 183
winfo_x() (*robot.libraries.dialogs_py.MessageDialog method*), 169
winfo_x() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 211
winfo_x() (*robot.libraries.dialogs_py.PassFailDialog method*), 225
winfo_x() (*robot.libraries.dialogs_py.SelectionDialog method*), 197
winfo_y() (*robot.libraries.dialogs_py.InputDialog method*), 183
winfo_y() (*robot.libraries.dialogs_py.MessageDialog method*), 169
winfo_y() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 211
winfo_y() (*robot.libraries.dialogs_py.PassFailDialog method*), 225
winfo_y() (*robot.libraries.dialogs_py.SelectionDialog method*), 197
WITH_NAME (*robot.parsing.lexer.tokens.END attribute*), 340
WITH_NAME (*robot.parsing.lexer.tokens.EOS attribute*), 337
WITH_NAME (*robot.parsing.lexer.tokens.Token attribute*), 334
with_traceback() (*robot.api.exceptions.ContinuableFailure*

method), 13
with_traceback() (*robot.api.exceptions.Error method*), 13
with_traceback() (*robot.api.exceptions.Failure method*), 12
with_traceback() (*robot.api.exceptions.FatalError method*), 13
with_traceback() (*robot.api.exceptions.SkipExecution method*), 14
with_traceback() (*robot.errors.BreakLoop method*), 618
with_traceback() (*robot.errors.ContinueLoop method*), 617
with_traceback() (*robot.errors.DataError method*), 613
with_traceback() (*robot.errors.ExecutionFailed method*), 615
with_traceback() (*robot.errors.ExecutionFailures method*), 616
with_traceback() (*robot.errors.ExecutionPassed method*), 616
with_traceback() (*robot.errors.ExecutionStatus method*), 615
with_traceback() (*robot.errors.FrameworkError method*), 613
with_traceback() (*robot.errors.HandlerExecutionFailed method*), 615
with_traceback() (*robot.errors.Information method*), 614
with_traceback() (*robot.errors.KeywordError method*), 614
with_traceback() (*robot.errors.PassExecution method*), 617
with_traceback() (*robot.errors.RemoteError method*), 618
with_traceback() (*robot.errors.ReturnFromKeyword method*), 618
with_traceback() (*robot.errors.RobotError method*), 613
with_traceback() (*robot.errors.TimeoutError method*), 614
with_traceback() (*robot.errors.UserKeywordExecutionFailed method*), 616
with_traceback() (*robot.errors.VariableError method*), 614
with_traceback() (*robot.libraries.BuiltIn.RobotNotRunningError method*), 99
with_traceback() (*robot.libraries.Telnet.NoMatchError method*), 146
withdraw() (*robot.libraries.dialogs_py.InputDialog method*), 183
withdraw() (*robot.libraries.dialogs_py.MessageDialog method*), 169
withdraw() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 211
withdraw() (*robot.libraries.dialogs_py.PassFailDialog method*), 225
withdraw() (*robot.libraries.dialogs_py.SelectionDialog method*), 197
wm_aspect() (*robot.libraries.dialogs_py.InputDialog method*), 183
wm_aspect() (*robot.libraries.dialogs_py.MessageDialog method*), 169
wm_aspect() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 211
wm_aspect() (*robot.libraries.dialogs_py.PassFailDialog method*), 225
wm_aspect() (*robot.libraries.dialogs_py.SelectionDialog method*), 197
wm_attributes() (*robot.libraries.dialogs_py.InputDialog method*), 183
wm_attributes() (*robot.libraries.dialogs_py.MessageDialog method*), 169
wm_attributes() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 211
wm_attributes() (*robot.libraries.dialogs_py.PassFailDialog method*), 225
wm_attributes() (*robot.libraries.dialogs_py.SelectionDialog method*), 197
wm_client() (*robot.libraries.dialogs_py.InputDialog method*), 183
wm_client() (*robot.libraries.dialogs_py.MessageDialog method*), 169
wm_client() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 211
wm_client() (*robot.libraries.dialogs_py.PassFailDialog method*), 225
wm_client() (*robot.libraries.dialogs_py.SelectionDialog method*), 197
wm_colormapwindows() (*robot.libraries.dialogs_py.InputDialog method*), 183
wm_colormapwindows() (*robot.libraries.dialogs_py.MessageDialog method*), 169
wm_colormapwindows() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 211
wm_colormapwindows() (*robot.libraries.dialogs_py.PassFailDialog method*), 225
wm_colormapwindows() (*robot.libraries.dialogs_py.SelectionDialog method*), 197
wm_command() (*robot.libraries.dialogs_py.InputDialog method*), 183
wm_command() (*robot.libraries.dialogs_py.MessageDialog method*), 169

wm_command() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 211
wm_command() (*robot.libraries.dialogs_py.PassFailDialog method*), 225
wm_command() (*robot.libraries.dialogs_py.SelectionDialog method*), 197
wm_deiconify() (*robot.libraries.dialogs_py.InputDialog method*), 184
wm_deiconify() (*robot.libraries.dialogs_py.MessageDialog method*), 170
wm_deiconify() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 212
wm_deiconify() (*robot.libraries.dialogs_py.PassFailDialog method*), 226
wm_deiconify() (*robot.libraries.dialogs_py.SelectionDialog method*), 198
wm_focusmodel() (*robot.libraries.dialogs_py.InputDialog method*), 184
wm_focusmodel() (*robot.libraries.dialogs_py.MessageDialog method*), 170
wm_focusmodel() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 212
wm_focusmodel() (*robot.libraries.dialogs_py.PassFailDialog method*), 226
wm_focusmodel() (*robot.libraries.dialogs_py.SelectionDialog method*), 198
wm_forget() (*robot.libraries.dialogs_py.InputDialog method*), 184
wm_forget() (*robot.libraries.dialogs_py.MessageDialog method*), 170
wm_forget() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 212
wm_forget() (*robot.libraries.dialogs_py.PassFailDialog method*), 226
wm_forget() (*robot.libraries.dialogs_py.SelectionDialog method*), 198
wm_frame() (*robot.libraries.dialogs_py.InputDialog method*), 184
wm_frame() (*robot.libraries.dialogs_py.MessageDialog method*), 170
wm_frame() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 212
wm_frame() (*robot.libraries.dialogs_py.PassFailDialog method*), 226
wm_frame() (*robot.libraries.dialogs_py.SelectionDialog method*), 198
wm_geometry() (*robot.libraries.dialogs_py.InputDialog method*), 184
wm_geometry() (*robot.libraries.dialogs_py.MessageDialog method*), 170
wm_geometry() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 212
wm_geometry() (*robot.libraries.dialogs_py.PassFailDialog method*), 226
wm_geometry() (*robot.libraries.dialogs_py.SelectionDialog method*), 198
wm_grid() (*robot.libraries.dialogs_py.InputDialog method*), 184
wm_grid() (*robot.libraries.dialogs_py.MessageDialog method*), 170
wm_grid() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 212
wm_grid() (*robot.libraries.dialogs_py.PassFailDialog method*), 226
wm_grid() (*robot.libraries.dialogs_py.SelectionDialog method*), 198
wm_group() (*robot.libraries.dialogs_py.InputDialog method*), 184
wm_group() (*robot.libraries.dialogs_py.MessageDialog method*), 170
wm_group() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 212
wm_group() (*robot.libraries.dialogs_py.PassFailDialog method*), 226
wm_group() (*robot.libraries.dialogs_py.SelectionDialog method*), 198
wm_iconbitmap() (*robot.libraries.dialogs_py.InputDialog method*), 184
wm_iconbitmap() (*robot.libraries.dialogs_py.MessageDialog method*), 170
wm_iconbitmap() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 212
wm_iconbitmap() (*robot.libraries.dialogs_py.PassFailDialog method*), 226
wm_iconbitmap() (*robot.libraries.dialogs_py.SelectionDialog method*), 198
wm_iconify() (*robot.libraries.dialogs_py.InputDialog method*), 184
wm_iconify() (*robot.libraries.dialogs_py.MessageDialog method*), 170
wm_iconify() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 212
wm_iconify() (*robot.libraries.dialogs_py.PassFailDialog method*), 226
wm_iconify() (*robot.libraries.dialogs_py.SelectionDialog method*), 198
wm_iconmask() (*robot.libraries.dialogs_py.InputDialog method*), 184
wm_iconmask() (*robot.libraries.dialogs_py.MessageDialog method*), 170
wm_iconmask() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 212
wm_iconmask() (*robot.libraries.dialogs_py.PassFailDialog method*), 226
wm_iconmask() (*robot.libraries.dialogs_py.SelectionDialog method*), 198
wm_iconname() (*robot.libraries.dialogs_py.InputDialog method*), 184

wm_iconname() (*robot.libraries.dialogs_py.MessageDialog method*), 170
wm_iconname() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 212
wm_iconname() (*robot.libraries.dialogs_py.PassFailDialog method*), 226
wm_iconname() (*robot.libraries.dialogs_py.SelectionDialog method*), 198
wm_iconphoto() (*robot.libraries.dialogs_py.InputDialog method*), 184
wm_iconphoto() (*robot.libraries.dialogs_py.MessageDialog method*), 170
wm_iconphoto() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 212
wm_iconphoto() (*robot.libraries.dialogs_py.PassFailDialog method*), 226
wm_iconphoto() (*robot.libraries.dialogs_py.SelectionDialog method*), 198
wm_iconposition() (*robot.libraries.dialogs_py.InputDialog method*), 185
wm_iconposition() (*robot.libraries.dialogs_py.MessageDialog method*), 171
wm_iconposition() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 213
wm_iconposition() (*robot.libraries.dialogs_py.PassFailDialog method*), 227
wm_iconposition() (*robot.libraries.dialogs_py.SelectionDialog method*), 199
wm_iconwindow() (*robot.libraries.dialogs_py.InputDialog method*), 185
wm_iconwindow() (*robot.libraries.dialogs_py.MessageDialog method*), 171
wm_iconwindow() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 213
wm_iconwindow() (*robot.libraries.dialogs_py.PassFailDialog method*), 227
wm_iconwindow() (*robot.libraries.dialogs_py.SelectionDialog method*), 199
wm_manage() (*robot.libraries.dialogs_py.InputDialog method*), 185
wm_manage() (*robot.libraries.dialogs_py.MessageDialog method*), 171
wm_manage() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 213
wm_manage() (*robot.libraries.dialogs_py.PassFailDialog method*), 227
wm_manage() (*robot.libraries.dialogs_py.SelectionDialog method*), 199
wm_maxsize() (*robot.libraries.dialogs_py.InputDialog method*), 185
wm_maxsize() (*robot.libraries.dialogs_py.MessageDialog method*), 171
wm_maxsize() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 213
wm_maxsize() (*robot.libraries.dialogs_py.PassFailDialog method*), 227
wm_maxsize() (*robot.libraries.dialogs_py.SelectionDialog method*), 199
wm_minsize() (*robot.libraries.dialogs_py.InputDialog method*), 185
wm_minsize() (*robot.libraries.dialogs_py.MessageDialog method*), 171
wm_minsize() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 213
wm_minsize() (*robot.libraries.dialogs_py.PassFailDialog method*), 227
wm_minsize() (*robot.libraries.dialogs_py.SelectionDialog method*), 199
wm_overrideredirect() (*robot.libraries.dialogs_py.InputDialog method*), 185
wm_overrideredirect() (*robot.libraries.dialogs_py.MessageDialog method*), 171
wm_overrideredirect() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 213
wm_overrideredirect() (*robot.libraries.dialogs_py.PassFailDialog method*), 227
wm_overrideredirect() (*robot.libraries.dialogs_py.SelectionDialog method*), 199
wm_positionfrom() (*robot.libraries.dialogs_py.InputDialog method*), 185
wm_positionfrom() (*robot.libraries.dialogs_py.MessageDialog method*), 171
wm_positionfrom() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 213
wm_positionfrom() (*robot.libraries.dialogs_py.PassFailDialog method*), 227
wm_positionfrom() (*robot.libraries.dialogs_py.SelectionDialog method*), 199
wm_protocol() (*robot.libraries.dialogs_py.InputDialog method*), 185
wm_protocol() (*robot.libraries.dialogs_py.MessageDialog method*), 171
wm_protocol() (*robot.libraries.dialogs_py.MultipleSelectionDialog*

method), 213
wm_protocol() (*robot.libraries.dialogs_py.PassFailDialog method*), 227
wm_protocol() (*robot.libraries.dialogs_py.SelectionDialog method*), 199
wm_resizable() (*robot.libraries.dialogs_py.InputDialog method*), 185
wm_resizable() (*robot.libraries.dialogs_py.MessageDialog method*), 171
wm_resizable() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 213
wm_resizable() (*robot.libraries.dialogs_py.PassFailDialog method*), 227
wm_resizable() (*robot.libraries.dialogs_py.SelectionDialog method*), 199
wm_sizefrom() (*robot.libraries.dialogs_py.InputDialog method*), 185
wm_sizefrom() (*robot.libraries.dialogs_py.MessageDialog method*), 171
wm_sizefrom() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 213
wm_sizefrom() (*robot.libraries.dialogs_py.PassFailDialog method*), 227
wm_sizefrom() (*robot.libraries.dialogs_py.SelectionDialog method*), 199
wm_state() (*robot.libraries.dialogs_py.InputDialog method*), 185
wm_state() (*robot.libraries.dialogs_py.MessageDialog method*), 171
wm_state() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 213
wm_state() (*robot.libraries.dialogs_py.PassFailDialog method*), 227
wm_state() (*robot.libraries.dialogs_py.SelectionDialog method*), 199
wm_title() (*robot.libraries.dialogs_py.InputDialog method*), 185
wm_title() (*robot.libraries.dialogs_py.MessageDialog method*), 171
wm_title() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 213
wm_title() (*robot.libraries.dialogs_py.PassFailDialog method*), 227
wm_title() (*robot.libraries.dialogs_py.SelectionDialog method*), 199
wm_transient() (*robot.libraries.dialogs_py.InputDialog method*), 185
wm_transient() (*robot.libraries.dialogs_py.MessageDialog method*), 171
wm_transient() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 213
wm_transient() (*robot.libraries.dialogs_py.PassFailDialog method*), 227
wm_transient() (*robot.libraries.dialogs_py.SelectionDialog method*), 199
wm_withdraw() (*robot.libraries.dialogs_py.InputDialog method*), 185
wm_withdraw() (*robot.libraries.dialogs_py.MessageDialog method*), 171
wm_withdraw() (*robot.libraries.dialogs_py.MultipleSelectionDialog method*), 213
wm_withdraw() (*robot.libraries.dialogs_py.PassFailDialog method*), 227
wm_withdraw() (*robot.libraries.dialogs_py.SelectionDialog method*), 199
write() (*in module robot.api.logger*), 15
write() (*in module robot.output.librarylogger*), 305
write() (*robot.htmldata.htmlfilewriter.CssFileWriter method*), 67
write() (*robot.htmldata.htmlfilewriter.GeneratorWriter method*), 67
write() (*robot.htmldata.htmlfilewriter.HtmlFileWriter method*), 67
write() (*robot.htmldata.htmlfilewriter.JsFileWriter method*), 67
write() (*robot.htmldata.htmlfilewriter.LineWriter method*), 67
write() (*robot.htmldata.htmlfilewriter.ModelWriter method*), 67
write() (*robot.htmldata.jsonwriter.JsonWriter method*), 68
write() (*robot.libdocpkg.htmlwriter.LibdocHtmlWriter method*), 71
write() (*robot.libdocpkg.htmlwriter.LibdocModelWriter method*), 71
write() (*robot.libdocpkg.jsonwriter.LibdocJsonWriter method*), 71
write() (*robot.libdocpkg.xmlwriter.LibdocXmlWriter method*), 73
write() (*robot.libraries.Telnet.TelnetConnection method*), 143
write() (*robot.output.console.highlighting.HighlightingStream method*), 303
write() (*robot.output.filelogger.FileLogger method*), 305
write() (*robot.output.logger.Logger method*), 309
write() (*robot.output.loggerhelper.AbstractLogger method*), 309
write() (*robot.output.output.Output method*), 311
write() (*robot.parsing.model.blocks.ModelWriter method*), 344
write() (*robot.reporting.jswriter.JsResultWriter method*), 395
write() (*robot.reporting.jswriter.SplitLogWriter method*), 395
write() (*robot.reporting.jswriter.SuiteWriter method*), 395
write() (*robot.reporting.logreportwriters.LogWriter*

method), 395
write() (*robot.reporting.logreportwriters.ReportWriter method*), 395
write() (*robot.reporting.logreportwriters.RobotModelWriter method*), 395
write() (*robot.reporting.xunitwriter.XUnitWriter method*), 402
write() (*robot.testdoc.TestdocModelWriter method*), 624
write_bare() (*robot.libraries.Telnet.TelnetConnection method*), 143
write_control_character() (*robot.libraries.Telnet.TelnetConnection method*), 143
write_data() (*robot.testdoc.TestdocModelWriter method*), 624
write_json() (*robot.htmldata.jsonwriter.JsonWriter method*), 68
write_results() (*robot.reporting.resultwriter.ResultWriter method*), 401
write_until_expected_output() (*robot.libraries.Telnet.TelnetConnection method*), 143

X

XML (*class in robot.libraries.XML*), 147
xml_escape() (*in module robot.utils.markuputils*), 602
XmlDocBuilder (*class in robot.libdocpkg.xmlbuilder*), 73
XmlElementHandler (*class in robot.result.xmlelementhandlers*), 519
XmlLogger (*class in robot.output.xmllogger*), 313
XmlRpcRemoteClient (*class in robot.libraries.Remote*), 129
XmlWriter (*class in robot.utils.markupwriters*), 602
xunit (*robot.conf.settings.RebotSettings attribute*), 67
xunit (*robot.conf.settings.RobotSettings attribute*), 66
XUnitFileWriter (*class in robot.reporting.xunitwriter*), 402
XUnitWriter (*class in robot.reporting.xunitwriter*), 402

Y

YamlImporter (*class in robot.variables.filesetter*), 607
yellow() (*robot.output.console.highlighting.AnsiHighlighter method*), 303
yellow() (*robot.output.console.highlighting.DosHighlighter method*), 304
yellow() (*robot.output.console.highlighting.NoHighlighting method*), 303

Z

ZhCn (*class in robot.conf.languages*), 53
ZhTw (*class in robot.conf.languages*), 55