# Introduction

Robot Framework is an **open-source Python-based** **automation framework for testing**.

The framework has a rich ecosystem with various libraries and tools developed as separate projects. For more information, see <http://robotframework.org>.

# Installation

<https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#installation-instructions>

$ pip install robotframework

# Test Approaches

There are several different ways in which test cases may be written.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Description** | **Example** | **Details** |
| **Keyword-Driven** | All tests contain a workflow constructed from keywords. | \*\*\* Test Cases \*\*\*  Push button      Push button    1      Result should be    1  Push multiple buttons      Push button    1      Push button    2      Result should be    12 | Keyword-driven and Gherkin-driven approaches are sometimes considered as workflow test.   * Generally have these phases:   + **Precondition** (optional – often in setup)   + **Action** (mandatory – do something to the system)   + **Verification** (mandatory – validate results)   + **Cleanup** (optional – always in teardown to make sure it is executed) * Different tests can have different abstraction levels.   + Tests for a detailed functionality are more precise.   + End-to-end tests can be on very high level.   + One test should use only one abstraction level * Different styles:   + More technical tests for lower level details and integration tests.   + "Executable specifications" **act as requirements**.   + Use *domain language* – Gherkin.   + Everyone should always understand. * **No complex logic** on the test case level.   + No for loops or if/else constructs.   + Use variable assignments with care.   + Test cases **should not look like scripts**! * **Max 10 steps**, preferably less. |
| **Gherkin-Driven**  **(Behavior-Driven)** | Similar to the keyword-driven, except that keywords use higher abstraction level and their arguments are embedded into the keyword names. | \*\*\* Test Cases \*\*\*  Addition      Given calculator has been cleared      When user types "1 + 1"      and user pushes equals      Then result is "2"  \*\*\* Keywords \*\*\*  Calculate      [Arguments]    ${expr}    ${expected}      Push buttons    C${expr}=      Result should be    ${expected} |
| **Data-Driven** | Tests use a keyword specified in the Test Template setting, that in turn uses other keywords.  This is very more suitable to writing testcases with same workflow repeated multiple times. | \*\*\* Test Cases \*\*\*    Expression    Expected  Addition              12 + 2 + 2    16                        2 + -3        -1  Subtraction           12 - 2 - 2    8                        2 - -3        5  Multiplication        12 \* 2 \* 2    48                        2 \* -3        -6 | * One high-level keyword per test.   + **Different arguments create different tests**.   + **One test can run the same keyword multiple times** to validate multiple related variations * If the keyword is implemented as a user keyword, it typically contains a similar workflow as workflow tests.   + Unless needed elsewhere, it is a good idea to create it in the same file as tests using it. * Recommended to use the *test template* functionality.   + No need to repeat the keyword multiple times.   + Easier to test multiple variations in one test. * Possible, and recommended, to name column headings * If a really big number of tests is needed, consider generating them based on an external model. |

# Quick Start

Find demostration in <https://github.com/robotframework/RobotDemo> or folder *Robot Framework/RobotDemo*.

To execute a specific test suite:

$ robot keyword\_driven.robot

To execute multiple test suites at once:

$ robot keyword\_driven.robot data\_driven.robot gherkin.robot

To execute all test suites in a directory recursively:

$ cd RobotDemo

$ robot .

# Writing Tests

## File Types

|  |  |
| --- | --- |
| **File Type** | **Description** |
| Test Suite | These files have .robot extension. **All** [**test cases**](#_Test_Case) **of a** [**test suite**](#_Test_Suite) are defined here.  This should be the first thing to look at to know what the test does. |
| Test Library | These are.py file. And [library](#_Test_Libraries) is just a Python class with methods that create **low-level keywords** used by the test cases. |
| Test Resource | [Resource files](#_Test_Resources) are .resource files, containing **variables** and **higher-level user keywords**. |
| Test Output | After running tests, you will **get different outputs**.   * **Output file**: Contains all execution results in machine readable XML or JSON format. Default file name is output.xml or output.json. * **Report file**: Contains an overview of the test execution results in HTML format. Default file name is log.html.   <https://robotframework.org/robotframework/latest/images/report_passed.html>  <https://robotframework.org/robotframework/latest/images/report_failed.html>   * **Log file**: Contains details about the executed test cases in HTML format. Default file name is report.html.   <https://robotframework.org/robotframework/latest/images/log_passed.html>  <https://robotframework.org/robotframework/latest/images/log_failed.html>  <https://robotframework.org/robotframework/latest/images/log_skipped.html>   * **Debug file**: Plain text files that are written during the test execution. All messages got from test libraries are written to them, as well as information about started and ended test suites, test cases and keywords. These files are not created by default, so must enable them with setting. * **XUnit compatible result file**: Contains the test execution summary in xUnit compatible XML format. Can be used as an input for external tools such as Jenkins. This file is not created by default, so must enable it with setting. |

## Spacing

There are two ways:

### Space Separated Format

Keywords and their arguments are separated from each others with **two or more spaces**. Or alternatively one or more tab characters.

The recommendation is using **4 spaces** for both indentaions and separations.

**\*\*\* Settings \*\*\***

Documentation     Example using the space separated format.

Library           OperatingSystem

**\*\*\* Variables \*\*\***

${MESSAGE}        Hello, world!

**\*\*\* Test Cases \*\*\***

My Test

    [Documentation]    Example test.

    Log    ${MESSAGE}

    My Keyword    ${CURDIR}

Another Test

    Should Be Equal    ${MESSAGE}    Hello, world!

**\*\*\* Keywords \*\*\***

My Keyword

    [Arguments]    ${path}

    Directory Should Exist    ${path}

### Pipe Separated Format

Keywords and their arguments are separated from each others with a pipe character surrounded with spaces (**|**).

| **\*\*\* Settings \*\*\***   |

| Documentation      | Example using the pipe separated format.

| Library            | OperatingSystem

| **\*\*\* Variables \*\*\***  |

| ${MESSAGE}         | Hello, world!

| **\*\*\* Test Cases \*\*\*** |                 |               |

| My Test            | [Documentation] | Example test. |

|                    | Log             | ${MESSAGE}    |

|                    | My Keyword      | ${CURDIR}     |

| Another Test       | Should Be Equal | ${MESSAGE}    | Hello, world!

| **\*\*\* Keywords \*\*\***   |                        |         |

| My Keyword         | [Arguments]            | ${path} |

|                    | Directory Should Exist | ${path} |

## Sections

Robot Framework data is defined in different sections, also called *tables*, listed below:

| **Section** | **Used for** |
| --- | --- |
| Settings | 1) Importing test libraries, resource files and variable files.  2) Defining metadata for test suites and test cases. |
| Variables | Defining variables that can be used elsewhere in the test data. |
| Test Cases | Creating test cases from available keywords. |
| Tasks | Creating tasks using available keywords. Single file can only contain either tests or tasks. |
| Keywords | Creating user keywords from existing lower-level keywords |
| Comments | Additional comments or data. Ignored by Robot Framework. |

Different sections are recognized by their header row. The **recommended header format is \*\*\* Settings \*\*\***, but the header is **case-insensitive**, surrounding **spaces are optional**, and the number of asterisk characters can vary as long as there is **at least one asterisk** in the beginning. For example, also \*settings would be recognized as a section header.

### *Settings* Section

#### Test Suite Settings

A **test suite** is described in the **Settings** section with following settings:

|  |  |
| --- | --- |
| **Setting** | **Used For** |
| [Name] | Used for setting a custom suite name. The default name is created based on the file or directory name.  Name is shown in reports and logs. |
| [Documentation] | Specify a test suite documentation. It is shown in reports and logs. |
| [Metadata] | Used for setting [free suite metadata](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#free-suite-metadata) as name-value pairs. It is shown in reports and logs. |
| [Suite Setup] | Specify test setup.  It is executed BEFORE running any of the suite's test cases or child test suites.  If it fails, all test cases in it and its child test suites are immediately assigned a fail status and ignore executed. |
| [Suite Teardown] | Specify test teardown.  It is executed AFTER running any of the suite's test cases or child test suites.  It is executed even if the setup of the same suite fails. If the suite teardown fails, all test cases in the suite are marked failed, regardless of their original execution status. |

Example:

**\*\*\* Settings \*\*\***

Name             Custom suite name

Documentation    An example suite documentation with \*some\* \_formatting\_.

...              Long documentation can be split into multiple lines.

Metadata         Version            2.0

Metadata         Robot Framework    http://robotframework.org

Metadata         Platform           ${PLATFORM}

Metadata         Longer Value

...              Longer metadata values can be split into multiple

...              rows. Also \*simple\* \_formatting\_ is supported.

#### Test Case Settings

**Global settings for test cases** are defined in the **Settings** section:

|  |  |
| --- | --- |
| **Setting** | **Used For** |
| [Test Setup] | Default values for test setup. |
| [Test Teardown] | Default values for test teardown. |
| [Test Tags] | Tags all tests in the suite will get in addition to their possible own tags. |
| [Test Template] | Default template keyword to use. |
| [Test Timeout] | Default value for test case timeout. |

Example:

**\*\*\* Settings \*\*\***

Name             Feature A

Documentation    Test suite for feature A

[Test Tags]     featureA

[Test Setup]     Configure logs

#### Test Library Importing

Test libraries must be imported using the Library setting in the **Settings** section and having the library name in the subsequent column. Unlike most of the other data, the library name is both **case- and space-sensitive**. If a library is in a package, the **full name** including the package name must be used.

It's possible for a library to **have arguments**. Both the library name and arguments can be set using variables.

Test libraries can be **imported in suite files or resource files**. All the keywords in the imported library are available in that file. With resource files, they are also available in other files using them.

Example 1: Library is a directory

**\*\*\* Settings \*\*\***

Library    OperatingSystem

Library    my.package.TestLibrary

Library    MyLibrary    arg1    arg2

Library    ${LIBRARY}

Example 2: Library is a file. The path is considered relative to the directory where current test data file is situated similarly as paths to resource and variable files.

**\*\*\* Settings \*\*\***

Library    PythonLibrary.py

Library    relative/path/PythonDirLib/    arg1    arg2

Library    ${RESOURCES}/Example.class

Example 3: Setting custom name to library. The name is shown in test logs before keyword names.

**\*\*\* Settings \*\*\***

Library    SomeLibrary    localhost        1234    AS    LocalLib

Library    SomeLibrary    server.domain    8080    AS    RemoteLib

#### Test Resource Importing

Resource files must be imported using the Resource setting in the **Settings** section so that the path to the resource file is given as an argument to the setting.

If the resource file path is absolute, it is used directly. Otherwise, the resource file is first searched relatively to the directory where the importing file is located. If the file is not found there, it is then searched from the directories in Python's module search path.

Example:

**\*\*\* Settings \*\*\***

Documentation     An example resource file

Library           SeleniumLibrary

Resource          ${RESOURCES}/common.resource

**\*\*\* Variables \*\*\***

${HOST}           localhost:7272

${LOGIN URL}      http://${HOST}/

${WELCOME URL}    http://${HOST}/welcome.html

${BROWSER}        Firefox

**\*\*\* Keywords \*\*\***

Open Login Page

    [Documentation]    Opens browser to login page

    Open Browser    ${LOGIN URL}    ${BROWSER}

    Title Should Be    Login Page

Input Name

    [Arguments]    ${name}

    Input Text    username\_field    ${name}

Input Password

    [Arguments]    ${password}

    Input Text    password\_field    ${password}

### *Test Cases* Section

Test cases are constructed in **Test Cases** sections from the available keywords.

Each test case has two parts: **name** (in the first column) and **steps** (in the second column). An exception to this rule is setting variables from keyword return values, when the second and possibly also the subsequent columns contain variable names and a keyword name is located after them. In either case, columns after the keyword name contain possible arguments to the specified keyword.

In addition, test cases can have their own **local** **settings**:

|  |  |
| --- | --- |
| **Setting** | **Used For** |
| [Documentation] | Specify a test case documentation. |
| [Setup] | Specify test setup. |
| [Teardown] | Specify test teardown. |
| [Template] | Specify the template keyword to use. The test itself will contain only data to use as arguments to that keyword. |
| [Tags] | Specify tag(s) for test cases.  They're helpful for **classifying test cases** and user keywords. Thus, they help to **collect statistics**, **select test to execute**, **remove duplicates**, etc. Note that tag comparisons are case-, space- and underscore-**insensitive**.  They are **free text** and Robot itself has no special meaning for them except for the reserved tags. |
| [Timeout] | Set a test case timeout. |

Example:

**\*\*\* Test Cases \*\*\***

Valid Login

    [Documentation]   Log in successfully with valid username and password

    [Tags]    normal

    Open Login Page

    Input Username    demo

    Input Password    mode

    Submit Credentials

    Welcome Page Should Be Open

Setting Variables

    Do Something    first argument    second argument

    ${value} =    Get Some Value

    Should Be Equal    ${value}    Expected value

### *Variables* Section

#### Variable Scope

##### Global

These variables are defined in the **Variables** section in suite files and resource files. However, note that values here are **always strings** (not [numbers](#_Number_Variables)) and they cannot be created dynamically. If either of these is a problem, [variable files](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#variable-files) can be used instead.

Example:

**\*\*\* Variables \*\*\***

${NAME}         Robot Framework

${VERSION}      2.0

${EMPTY}

${ROBOT}        ${NAME} ${VERSION}

${MULTILINE}    First line.

...             Second line.

...             Third line.

...             separator=\n

@{MANY}         one         two      three      four

&{USER}       name=Matti    address=xxx         phone=123

It is also possible, but not obligatory, to use the equals sign **=** after the variable name to make assigning variables slightly more explicit.:

\*\*\* Variables \*\*\*

${NAME} =       Robot Framework

${VERSION} =    2.0

##### Local

**Return values from keywords** can be set into variables. This allows communication between different keywords.

Any value returned by a keyword can be assigned to a scalar variable. Having the equals sign **=** after the variable name is not obligatory, but it makes the assignment more explicit:

**\*\*\* Test Cases \*\*\***

Returning

    ${x} =    Get X    an argument

    Log    We got ${x}!

Although a value is assigned to a scalar variable, it can be used as a list variable if it has a list-like value and as a dictionary variable if it has a dictionary-like value:

**\*\*\* Test Cases \*\*\***

Example

    ${list} =    Create List    first    second    third

    Length Should Be    ${list}    3

    Log Many    @{list}

**You might not know!**

Variable scope can be changed dynamically:

- Way 1: <https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#toc-entry-332>

- Way 2 (recommended): <https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#toc-entry-331>

#### Data Types

|  |  |
| --- | --- |
| **Type** | **Syntax** |
| Scalar | ${SCALAR} |
| List | @{LIST} |
| Dictionary | &{DICT} |
| Environment Variable | %{ENV\_VAR} |

##### Scalar

###### String

**\*\*\* Variables \*\*\***

${GREET}         Hello

${NAME}       World

**\*\*\* Test Cases \*\*\***

Constants

    Log    Hello

    Log    Hello, world!!

Variables

    Log    ${GREET}                # Hello

    Log    ${GREET}, ${NAME}!!     # Hello, World!!

When a scalar variable is used alone, like in ${GREET} above, it is replaced with its value as-is and the value can be any object.

If the variable is not used alone, but having other text or variables around, like ${GREET}, ${NAME}!! above, its value is first converted into a string and then concatenated with the other data.

###### Number

**\*\*\* Test Cases \*\*\***

Example 1A

    Connect    example.com    80       # Connect gets two strings as arguments

Example 1B

    Connect    example.com    ${80}    # Connect gets a string and an integer

Example 2

    Do X    ${3.14}    ${-1e-4}        # Do X gets floating point numbers 3.14 and -0.0001

It is possible to create also from binary, octal, and hexadecimal values using 0b, 0o and 0x prefixes, respectively:

**\*\*\* Test Cases \*\*\***

Example

    Should Be Equal    ${0b1011}    ${11}

    Should Be Equal    ${0o10}      ${8}

    Should Be Equal    ${0xff}      ${255}

    Should Be Equal    ${0B1010}    ${0XA}

###### Boolean

**\*\*\* Test Cases \*\*\***

Boolean

    Set Status    ${true}               # Set Status gets boolean "true" as an argument

    Create Y    something   ${false}    # Create Y gets a string and boolean "false"

###### None/null

**\*\*\* Test Cases \*\*\***

None

    Do XYZ    ${None}                   # Do XYZ gets Python None as an argument

##### List

The following test cases, Constants and List Variables, are equivalent:

**\*\*\* Variables \*\*\***

@{USER} robot secret

**\*\*\* Test Cases \*\*\***

Constants

    Login    robot    secret

List Variable

    Login    @{USER}

Also, *list expansion* can be used in combination with [list item access](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#accessing-sequence-items) making these usages possible:

**\*\*\* Test Cases \*\*\***

Nested container

    ${nested} =    Evaluate    [['a', 'b', 'c'], {'key': ['x', 'y']}]

    Log Many    @{nested}[0]         # Logs 'a', 'b' and 'c'.

    Log Many    @{nested}[1][key]    # Logs 'x' and 'y'.

Slice

    ${items} =    Create List    first    second    third

    Log Many    @{items}[1:]         # Logs 'second' and  'third'.

##### Dict

The following test cases, Constants and Dict Variable, are equivalent:

**\*\*\* Variables \*\*\***

&{USER} name=robot password=secret

**\*\*\* Test Cases \*\*\***

Constants

    Login    name=robot    password=secret

Dict Variable

    Login    &{USER}

A certain value of a dictionary variable can be accessed with the syntax ${NAME}[key], where key is the name of the selected value. Keys are considered to be strings, but non-strings keys can be used as variables. If a key is a string, it is possible to access its value also using attribute access syntax ${NAME.key}.

**\*\*\* Variables \*\*\***

&{USER} name=robot password=secret

**\*\*\* Test Cases \*\*\***

Dictionary variable item

    Login    ${USER}[name]    ${USER}[password]

    Title Should Be    Welcome ${USER}[name]!

Key defined as variable

    Log Many    ${DICT}[${KEY}]    ${DICT}[${42}]

Attribute access

    Login    ${USER.name}    ${USER.password}

    Title Should Be    Welcome ${USER.name}!

##### Env Var

Env vars are limited to string values. It is possible to specify a default value, that is used if the env var does not exists, by separating the variable name and the default value with an equal sign like %{ENV\_VAR\_NAME=default value}.

Env vars, which is set in the OS before the test execution, are available during it, and it's possible to create new ones with the keyword Set Environment Variable or delete existing ones with the keyword Delete Environment Variable, both available in the OperatingSystem library. Because env vars are global, ones set in one test case can be used in other test cases executed after it. However, changes to env vars are not effective after the test execution.

#### Built-In Variables

Robot Framework provides some built-in variables that are available automatically.

##### OS

Built-in variables related to the operating system ease making the test data operating-system-agnostic.

| **Variable** | **Explanation** |
| --- | --- |
| ${CURDIR} | Absolute path to the directory where the test data file is located. This variable is case-sensitive. |
| ${TEMPDIR} | Absolute path to the system temporary directory.  In UNIX-like systems this is typically */tmp*, and in Windows *c:\Documents and Settings\<user>\Local Settings\Temp*. |
| ${EXECDIR} | Absolute path to the directory where test execution was started from. |
| ${/} | System directory path separator. / in UNIX-like systems and \ in Windows. |
| ${:} | System path element separator. : in UNIX-like systems and ; in Windows. |
| ${\n} | System line separator. \n in UNIX-like systems and \r\n in Windows. |

##### Testcase

| **Variable** | **Explanation** | **Available** |
| --- | --- | --- |
| ${TEST NAME} | The name of the current test case. | Test case |
| @{TEST TAGS} | Contains the tags of the current test case in alphabetical order. Can be modified dynamically using *Set Tags* and *Remove Tags* keywords. | Test case |
| ${TEST DOCUMENTATION} | The documentation of the current test case. Can be set dynamically using using *Set Test Documentation* keyword. | Test case |
| ${TEST STATUS} | The status of the current test case, either PASS or FAIL. | [Test teardown](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#test-setup-and-teardown) |
| ${TEST MESSAGE} | The message of the current test case. | [Test teardown](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#test-setup-and-teardown) |
| ${PREV TEST NAME} | The name of the previous test case, or an empty string if no tests have been executed yet. | Everywhere |
| ${PREV TEST STATUS} | The status of the previous test case: either PASS, FAIL, or an empty string when no tests have been executed. | Everywhere |
| ${PREV TEST MESSAGE} | The possible error message of the previous test case. | Everywhere |
| ${SUITE NAME} | The full name of the current test suite. | Everywhere |
| ${SUITE SOURCE} | An absolute path to the suite file or directory. | Everywhere |
| ${SUITE DOCUMENTATION} | The documentation of the current test suite. Can be set dynamically using using *Set Suite Documentation* keyword. | Everywhere |
| &{SUITE METADATA} | The free metadata of the current test suite. Can be set using *Set Suite Metadata* keyword. | Everywhere |
| ${SUITE STATUS} | The status of the current test suite, either PASS or FAIL. | [Suite teardown](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#suite-setup-and-teardown) |
| ${SUITE MESSAGE} | The full message of the current test suite, including statistics. | [Suite teardown](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#suite-setup-and-teardown) |
| ${KEYWORD STATUS} | The status of the current keyword, either PASS or FAIL. | [User keyword teardown](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#user-keyword-teardown) |
| ${KEYWORD MESSAGE} | The possible error message of the current keyword. | [User keyword teardown](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#user-keyword-teardown) |
| ${LOG LEVEL} | Current [log level](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#log-levels). | Everywhere |
| ${OUTPUT DIR} | An absolute path to the [output directory](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#output-directory) as a string. | Everywhere |
| ${OUTPUT FILE} | An absolute path to the [output file](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#output-file) as a string or a string NONE if the output file is not created. | Everywhere |
| ${LOG FILE} | An absolute path to the [log file](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#log-file) as a string or a string NONE if the log file is not created. | Everywhere |
| ${REPORT FILE} | An absolute path to the [report file](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#report-file) as a string or a string NONE if the report file is not created. | Everywhere |
| ${DEBUG FILE} | An absolute path to the [debug file](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#debug-file) as a string or a string NONE if the debug file is not created. | Everywhere |
| &{OPTIONS} | A dictionary exposing command line options. The dictionary keys match the command line options and can be accessed both like ${OPTIONS}[key] and ${OPTIONS.key}. Available options:   * ${OPTIONS.exclude} (--exclude) * ${OPTIONS.include} (--include) * ${OPTIONS.skip} (--skip) * ${OPTIONS.skip\_on\_failure} (--skip-on-failure) * ${OPTIONS.console\_width} (--console-width)   ${OPTIONS} itself was added in RF 5.0 and ${OPTIONS.console\_width} in RF 7.1. More options can be exposed later. | Everyw |

### *Keywords* Section

Keywords are created in the **Keywords** section in suite files and resource files. They can be user-defined or from library.

In this example, we have 5 keywords: Open Login Page, Open Browser, URL Should Start With, Get URL, and Should Start With.

**\*\*\* Keywords \*\*\***

Open Login Page

    Open Browser    http://host/login.html

URL Should Start With

    ${url} =    Get URL

    Should Start With    ${url}    http

#### Keyword Settings

Keywords can have settings, just like test cases:

|  |  |
| --- | --- |
| **Setting** | **Used For** |
| [Documentation] | Specify a keyword documentation. |
| [Setup] | Specify test setup.  Unlike tes tcases, it is not possible to specify a common setup to all keywords in a certain file. |
| [Teardown] | Specify test teardown.  Unlike tes tcases, it is not possible to specify a common teardown to all keywords in a certain file. |
| [Arguments] | Specify the keyword arguments. |
| [Return] | Specifies keyword return values  Deprecated in Robot Framework 7.0, the RETURN statement should be used instead. |
| [Tags] | Tag keyword. |
| [Timeout] | Set a keyword timeout. |

#### Keyword Arguments

Keywords can have zero or more arguments, and some arguments may have default values. It is also possible that a keyword accepts any number of arguments.

**\*\*\* Test Cases \*\*\***

Example

    No Operation

    Create Directory    ${TEMPDIR}/stuff

    Copy File    ${CURDIR}/file.txt    ${TEMPDIR}/stuff

    Create File    ${TEMPDIR}/empty.txt

    Create File    ${TEMPDIR}/utf-8.txt         content=Here content is not empty

    Create File    ${TEMPDIR}/iso-8859-1.txt    Here content is not empty with encoding    encoding=ISO-8859-1

Remove Files ${TEMPDIR}/f1.txt ${TEMPDIR}/f2.txt ${TEMPDIR}/f3.txt

Note: Above keywords are built-in in [OperatingSystem](https://robotframework.org/robotframework/latest/libraries/OperatingSystem.html) library.

Example:

**\*\*\* Keywords \*\*\***

One Argument

    [Documentation]    This is a keyword with one argument

    [Arguments]    ${arg\_name}

    Log    Got argument ${arg\_name}

Three Arguments

    [Tags]  Example

    [Arguments]    ${arg1}    ${arg2}    ${arg3}

    Log    1st argument: ${arg1}

    Log    2nd argument: ${arg2}

    Log    3rd argument: ${arg3}

One Argument With Default Value

    [Arguments]    ${arg}=default value

    Log    Got argument ${arg}

One Required And One With Default

    [Arguments]    ${required}    ${optional}=default

    Log    Required: ${required}

    Log    Optional: ${optional}

Return One Value

    [Arguments]    ${arg}

    [Documentation]    Return a value unconditionally.

    ...                Notice that keywords after RETURN are not executed.

    ${value} =    Convert To Upper Case    ${arg}

    RETURN    ${value}

    Fail    Not executed

Return Three Values

    RETURN    a    b    c

Return One Value (Old style)

    [Arguments]    ${arg}

    ${value} =    Convert To Upper Case    ${arg}

    [Return]    ${value}

Setup and teardown

    [Setup]       Log    New in RF 7!

    Do Something

    [Teardown]    Log    Old feature.

Using variables

    [Setup]       ${SETUP}

    Do Something

    [Teardown]    ${TEARDOWN}

#### Embedding Arguments Into Keyword Name

**\*\*\* Test Cases \*\*\***

Embedded arguments

    Select cat from list

    Select dog from list

**\*\*\* Keywords \*\*\***

Select ${animal} from list

    Open Page    Pet Selection

    Select Item From List    animal\_list    ${animal}

Number of ${animals} should be

    [Arguments]    ${count}

    Open Page    Pet Selection

    Select Items From List    animal\_list    ${animals}

    Number of Selected List Items Should Be    ${count}

**Notes:**

* Embedded arguments do not support default values or variable number of arguments like normal arguments do.
* Embedded arguments can cause wrong-value matching. An easy solution to this problem is surrounding arguments with double quotes. For example, without double quotes, ${city} will contain Los and ${team} will contain Angeles Lakers, which is wrong:

**\*\*\* Test Cases \*\*\***

Example

Select "Chicago" "Bulls"

Select "Los Angeles" "Lakers"

**\*\*\* Keywords \*\*\***

Select "${city}" "${team}"

Log Selected ${team} from ${city}.    Number of Selected List Items Should Be    ${count}

#### Keyword Return Value

The recommended approach is using the native RETURN statement. The old [Return] setting was deprecated and also built-in Return From Keyword and Return From Keyword keyword is considered deprecated.

**\*\*\* Settings \*\*\***

Library String

**\*\*\* Test Cases \*\*\***

One return value

${ret} = Return One Value argument

Should Be Equal ${ret} ARGUMENT

Multiple return values

${a} ${b} ${c} = Return Three Values

Should Be Equal ${a}, ${b}, ${c} a, b, c

Conditional return

Conditional Return 1

Conditional Return 2

Advanced

@{list} = Create List foo bar baz

${index} = Find Index bar ${list}

Should Be Equal ${index} ${1}

${index} = Find Index non existing ${list}

Should Be Equal ${index} ${-1}

**\*\*\* Keywords \*\*\***

Return One Value

[Arguments] ${arg}

[Documentation] Return a value unconditionally.

... Notice that keywords after RETURN are not executed.

${value} = Convert To Upper Case ${arg}

RETURN ${value}

Fail Not executed

Return Three Values

[Documentation] Return multiple values.

RETURN a b c

Conditional Return

[Arguments] ${arg}

[Documentation] Return conditionally.

Log Before

IF ${arg} == 1

Log Returning!

RETURN

END

Log After

Find Index

[Arguments] ${test} ${items}

[Documentation] Advanced example involving FOR loop, inline IF and @{list} variable syntax.

FOR ${index} ${item} IN ENUMERATE @{items}

IF $item == $test RETURN ${index}

END

RETURN ${-1}

### *Tasks* Section

### *Comments* Section

## Test Setup and Teardown

A test setup is executed before a test case, and a test teardown is executed after a test case. In Robot Framework, setups and teardowns are just normal keywords with possible arguments.

**\*\*\* Settings \*\*\***

Test Setup       Open Application    App A

Test Teardown    Close Application

**\*\*\* Test Cases \*\*\***

Default values

    [Documentation]    Setup and teardown from setting section

    Do Something

Overridden setup

    [Documentation]    Own setup, teardown from setting section

    [Setup]    Open Application    App B

    Do Something

No teardown

    [Documentation]    Default setup, no teardown at all

    Do Something

    [Teardown]

No teardown 2

    [Documentation]    Setup and teardown can be disabled also with special value NONE

    Do Something

    [Teardown]    NONE

Using variables

    [Documentation]    Setup and teardown specified using variables

    [Setup]    ${SETUP}

    Do Something

    [Teardown]    ${TEARDOWN}

## Test Templates

Test templates **convert normal keyword-driven test cases into data-driven tests**. Whereas the body of a keyword-driven test case is constructed from keywords and their possible arguments, test cases with template contain only the arguments for the template keyword. Instead of repeating the same keyword multiple times per test and/or with all tests in a file, it is possible to use it only per test or just once per file.

### Simple Example

Specify the template for an individual test case using the [Template] setting.

Below two tests are functionally fully identical:

**\*\*\* Test Cases \*\*\***

Normal test case

    Example keyword    first argument    second argument

Templated test case

    [Template]    Example keyword

    first argument    second argument

### More Complex Example

The below example has six separate tests, one for each invalid user/password combination, and the example below **illustrates how to have only one test with all the combinations**. This is where we can see the benefits of using test template:

**\*\*\* Settings \*\*\***

Test Template    Login with invalid credentials should fail

**\*\*\* Test Cases \*\*\***                USERNAME         PASSWORD

Invalid User Name                 invalid          ${VALID PASSWORD}

Invalid Password                  ${VALID USER}    invalid

Invalid User Name and Password    invalid          invalid

Empty User Name                   ${EMPTY}         ${VALID PASSWORD}

Empty Password                    ${VALID USER}    ${EMPTY}

Empty User Name and Password      ${EMPTY}         ${EMPTY}

## Test Libraries

### Standard Library

<https://robotframework.org/robotframework/#standard-libraries>

|  |  |
| --- | --- |
| **Library** | **Introduction** |
| BuiltIn | Contains generic often needed keywords. Imported automatically and thus always available. |
| Collections | Contains keywords for handling lists and dictionaries. |
| DateTime | Supports creating and verifying date and time values as well as calculations between them. |
| Dialogs | Supports pausing the test execution and getting input from users. |
| OperatingSystem | Enables performing various operating system related tasks. |
| Process | Supports executing processes in the system. |
| Screenshot | Provides keywords to capture and store screenshots of the desktop. |
| String | Library for manipulating strings and verifying their contents. |
| Telnet | Supports connecting to Telnet servers and executing commands on the opened connections. |
| XML | Library for verifying and modifying XML documents. |

### Remote Library

In addition to the standard libraries listed above, there is Remote library that is totally different than the other standard libraries. It does not have any keywords of its own but it works as a proxy between Robot Framework and actual test library implementations. These libraries can be running on other machines than the core framework and can even be implemented using languages not supported by Robot Framework natively.

<https://github.com/robotframework/RemoteInterface>

### External Library

Any test library that is **not one of the standard libraries** is an external library. The Robot Framework open source community has implemented several generic libraries, such as SeleniumLibrary and SwingLibrary, which are not packaged with the core framework. A list of publicly available libraries can be found from <http://robotframework.org>.

## Control Structures

Various structures can be used to **control the test execution flow**. These are **familiar** from most programming languages as they allow conditional execution, repeatedly executing a block of keywords and fine-grained error handling.

For readability reasons these structures should be used judiciously, and more complex use cases should be preferably implemented in test libraries.

### Loops

#### FOR Loops

**\*\*\* Test Cases \*\*\***

Simple for loop

    FOR    ${animal}    IN    cat    dog

        Log    ${animal}

        Log    2nd keyword

    END

    Log    Outside loop

Another simple for loop

    FOR    ${var}    IN    one    two    ${3}    four    ${five}

    ...    kuusi    7    eight    nine    ${last}

        Log    ${var}

    END

Nested for loops

    [Arguments]    @{table}

    FOR    ${row}    IN    @{table}

        FOR    ${cell}    IN    @{row}

            Handle Cell    ${cell}

        END

    END

#### FOR-IN-RANGE Loop

**\*\*\* Test Cases \*\*\***

Only upper limit

    [Documentation]    Loops over values from 0 to 9.

    FOR    ${index}    IN RANGE    10

        Log    ${index}

    END

Start and end

    [Documentation]    Loops over values from 1 to 10.

    FOR    ${index}    IN RANGE    1    11

        Log    ${index}

    END

Also step given

    [Documentation]    Loops over values 5, 15, and 25.

    FOR    ${index}    IN RANGE    5    26    10

        Log    ${index}

    END

Negative step

    [Documentation]    Loops over values 13, 3, and -7.

    FOR    ${index}    IN RANGE    13    -13    -10

        Log    ${index}

    END

Arithmetic

    [Documentation]    Arithmetic with variable.

    FOR    ${index}    IN RANGE    ${var} + 1

        Log    ${index}

    END

Float parameters

    [Documentation]    Loops over values 3.14, 4.34, and 5.54.

    FOR    ${index}    IN RANGE    3.14    6.09    1.2

        Log    ${index}

    END

#### WHILE Loops

**\*\*\* Test Cases \*\*\***

Example

    VAR    ${rc}   1

    WHILE    ${rc} != 0

        ${rc} =    Keyword that returns zero on success

    END

Limit as iteration count

    WHILE    True    limit=100

        Log    This is run 100 times.

    END

    WHILE    True    limit=10 times

        Log    This is run 10 times.

    END

    WHILE    True    limit=42x

        Log    This is run 42 times.

    END

Limit as time

    WHILE    True    limit=10 seconds

        Log    This is run 10 seconds.

    END

No limit

    WHILE    True    limit=NONE

        Log    This runs forever.

    END

#### BREAK and CONTINUE in Loops

**\*\*\* Test Cases \*\*\***

BREAK with FOR

    ${text} =    Set Variable    zero

    FOR    ${var}    IN    one    two    three

        IF    '${var}' == 'two'    BREAK

        ${text} =    Set Variable    ${text}-${var}

    END

    Should Be Equal    ${text}    zero-one

CONTINUE with FOR

    ${text} =    Set Variable    zero

    FOR    ${var}    IN    one    two    three

        IF    '${var}' == 'two'    CONTINUE

        ${text} =    Set Variable    ${text}-${var}

    END

    Should Be Equal    ${text}    zero-one-three

### IF/ELSE Condition

**\*\*\* Test Cases \*\*\***

IF

   IF    ${rc} > 0

       Some keyword

       Another keyword

   END

IF ELSE

    IF    ${rc} > 0

        Some keyword

    ELSE

        Another keyword

    END

IF ELSE IF ELSE

    IF    $rc > 0

        Positive keyword

    ELSE IF    $rc < 0

        Negative keyword

    ELSE IF    $rc == 0

        Zero keyword

    ELSE

        Fail    Unexpected rc: ${rc}

    END

Inline IF

    IF    $condition1    Keyword    argument

    IF    $condition2    RETURN

Inline IF/ELSE

    IF    $condition    Keyword    argument    ELSE    Another Keyword

Inline IF/ELSE IF/ELSE

    IF    $cond1    Keyword 1    ELSE IF    $cond2    Keyword 2    ELSE IF    $cond3    Keyword 3    ELSE    Keyword 4

Nested IF/ELSE

    [Arguments]    @{items}    ${log\_values}=True

    IF    not ${items}

        Log to console    No items.

    ELSE IF    len(${items}) == 1

        IF    ${log\_values}

            Log to console    One item: ${items}[0]

        ELSE

            Log to console    One item.

        END

    ELSE

        Log to console    ${{len(${items})}} items.

        IF    ${log\_values}

            FOR    ${index}    ${item}    IN ENUMERATE    @{items}    start=1

                Log to console    Item ${index}: ${item}

            END

        END

    END

### TRY/EXCEPT Syntax

**\*\*\* Test Cases \*\*\***

First example

    TRY

        Some Keyword

    EXCEPT    Error message

        Error Handler Keyword

    END

    Keyword Outside

Multiple EXCEPT branches

    TRY

        Some Keyword

    EXCEPT    Error message    # Try matching this first.

        Error Handler 1

    EXCEPT    Another error    # Try this if above did not match.

        Error Handler 2

    EXCEPT    ${message}       # Last match attempt, this time using a variable.

        Error Handler 3

    END

Multiple messages with one EXCEPT

    TRY

        Some Keyword

    EXCEPT    Error message    Another error    ${message}    # Match any of these.

        Error handler

    END

Match any error

    TRY

        Some Keyword

    EXCEPT               # Match any error.

        Error Handler

    END

Match any after testing more specific errors

    TRY

        Some Keyword

    EXCEPT    Error message    # Try matching this first

        Error Handler 1

    EXCEPT                     # Match any that did not match the above.

        Error Handler 2

    END

TRY/EXCEPT/ELSE/FINALLY

    TRY

        Some keyword

    EXCEPT

        Log    Error occurred!

    ELSE

        Log    No error occurred.

    FINALLY

        Log    Always executed.

    END

TRY/FINALLY

    Open Connection

    TRY

        Use Connection

    FINALLY

        Close Connection

    END

## Variable Files

Variable files contain variables used in the test data. Variables can also be created using Variable sections or set from the command line, but variable files allow creating them **dynamically** and with **different data types** (not only strings).

There are many ways to create variable files: <https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#variable-files>

But the most simple and helpful is using YAML files.

For example:

|  |  |
| --- | --- |
| Simple YAML file named common/yaml:  string: Hello, world!  integer: 42  list:  - one  - two  dict:  one: yksi  two: kaksi  with spaces: kolme | If the above YAML file is imported, it will create exactly the same variables as the Variable section:  **\*\*\* Settings \*\*\***  Variables ${RESOURCES}/common.yaml  **\*\*\* Variables \*\*\***  ${STRING} Hello, world!  ${INTEGER} ${42}  @{LIST} one two  &{DICT} one=yksi two=kaksi with spaces=kolme |

## Timeout

<https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#timeouts>

### Test Case Timeout

### Keyword Timeout

# Executing Tests

<https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#executing-test-cases>

## CLI

|  |  |
| --- | --- |
| **Option** | **Description** |
| --rpa | Turn on [generic automation](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#task-execution) mode. |
| --language <lang> | Activate [localization](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#localization).  lang can be a name or a code of a [built-in language](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#translations), or a path or a module name of a custom language file. |
| -F, --extension <value> | [Parse only these files](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#selecting-files-to-parse) when executing a directory. |
| -I, --parseinclude <pattern> | [Parse only matching files](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#selecting-files-to-parse) when executing a directory. |
| -N, --name <name> | [Sets the name](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#setting-suite-name) of the top-level test suite. |
| -D, --doc <document> | [Sets the documentation](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#setting-suite-documentation) of the top-level test suite. |
| -M, --metadata <name:value> | [Sets free metadata](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#setting-free-suite-metadata) for the top level test suite. |
| -G, --settag <tag> | [Sets the tag(s)](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#setting-test-tags) to all executed test cases. |
| -t, --test <name> | [Selects the test cases by name](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#by-test-names). |
| --task <name> | Alias for --test that can be used when [executing tasks](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#executing-tasks). |
| -s, --suite <name> | [Selects the test suites](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#by-suite-names) by name. |
| -R, --rerunfailed <file> | [Selects failed tests](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#re-executing-failed-test-cases) from an earlier [output file](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#output-file) to be re-executed. |
| -S, --rerunfailedsuites <file> | [Selects failed test suites](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#re-executing-failed-test-suites) from an earlier [output file](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#output-file) to be re-executed. |
| -i, --include <tag> | [Selects the test cases](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#by-tag-names) by tag. |
| -e, --exclude <tag> | [Selects the test cases](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#by-tag-names) by tag. |
| --skip <tag> | Tests having given tag will be [skipped](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#skipped). Tag can be a pattern. |
| --skiponfailure <tag> | Tests having given tag will be [skipped](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#skipped) if they fail. |
| -v, --variable <name:value> | Sets [individual variables](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#setting-variables-in-command-line). |
| -V, --variablefile <path:args> | Sets variables using [variable files](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#variable-files). |
| -d, --outputdir <dir> | Defines where to [create output files](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#output-directory) – aka *output directory*.  The path is relative to the execution directory, but can be given as an absolute path. |
| -o, --output <file> | Sets the path to the generated [output file](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#output-file) in XML or JSON format.  The path is relative to the *output directory* and the default value is output.xml when executing tests.  It's possible to disable the output file by setting --output to NONE. |
| --legacyoutput | Creates output file in [Robot Framework 6.x compatible format](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#legacy-xml-format). |
| -l, --log <file> | Sets the path to the generated [log file](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#log-file).  Default value is log.html  It's possible to disable the log file by setting --log to NONE. |
| -r, --report <file> | Sets the path to the generated [report file](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#report-file).  Default value is report.html  It's possible to disable the report file by setting --report to NONE. |
| -x, --xunit <file> | Sets the path to the generated [xUnit compatible result file](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#xunit-compatible-result-file). |
| -b, --debugfile <file> | A [debug file](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#debug-file) that is written during execution. |
| -T, --timestampoutputs | [Adds a timestamp](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#timestamping-output-files) to [output files](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#output-file) listed above. |
| --splitlog | [Split log file](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#splitting-logs) into smaller pieces that open in browser transparently. |
| --logtitle <title> | [Sets a title](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#setting-titles) for the generated test log. |
| --reporttitle <title> | [Sets a title](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#setting-titles) for the generated test report. |
| --reportbackground <colors> | [Sets background colors](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#setting-background-colors) of the generated report. |
| --maxerrorlines <lines> | Sets the number of [error lines](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#limiting-error-message-length-in-reports) shown in report when tests fail. |
| --maxassignlength <characters> | Sets the number of characters shown in log when [variables are assigned](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#automatically-logging-assigned-variable-value). |
| -L, --loglevel <level> | [Sets the threshold level](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#setting-log-level) for logging. Optionally the default [visible log level](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#visible-log-level) can be given separated with a colon (:). |
| --suitestatlevel <level> | Defines how many [levels to show](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#configuring-displayed-suite-statistics) in the *Statistics by Suite* table in outputs. |
| --tagstatinclude <tag> | [Includes only these tags](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#including-and-excluding-tag-statistics) in the *Statistics by Tag* table. |
| --tagstatexclude <tag> | [Excludes these tags](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#including-and-excluding-tag-statistics) from the *Statistics by Tag* table. |
| --tagstatcombine <tags:title> | Creates [combined statistics based on tags](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#generating-combined-tag-statistics). |
| --tagdoc <pattern:doc> | Adds [documentation to the specified tags](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#adding-documentation-to-tags). |
| --tagstatlink <pattern:link:title> | Adds [external links](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#creating-links-from-tag-names) to the *Statistics by Tag* table. |
| --expandkeywords <name:pattern|tag:pattern> | Automatically [expand keywords](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#automatically-expanding-keywords) in the generated log file. |
| --removekeywords  <all|passed|name:pattern|tag:pattern|for|while|wuks> | [Removes keyword data](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#removing-and-flattening-keywords) from the generated log file. |
| --flattenkeywords  <for|while|iteration|name:pattern|tag:pattern> | [Flattens keywords](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#removing-and-flattening-keywords) in the generated log file. |
| --listener <name:args> | [Sets a listener](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#setting-listeners) for monitoring test execution. |
| --nostatusrc | Sets the [return code](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#return-codes) to zero regardless of failures in test cases. Error codes are returned normally. |
| --runemptysuite | Executes tests also if the selected [test suites are empty](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#when-no-tests-match-selection). |
| --dryrun | In the [dry run](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#dry-run) mode, tests are run without executing keywords originating from test libraries.  Useful for validating test data syntax. |
| -X, --exitonfailure | [Stops test execution](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#stopping-when-first-test-case-fails) if any test fails. |
| --exitonerror | [Stops test execution](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#stopping-on-parsing-or-execution-error) if any error occurs when parsing test data, importing libraries, and so on. |
| --skipteardownonexit | [Skips teardowns](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#handling-teardowns) if test execution is prematurely stopped. |
| --prerunmodifier <name:args> | Activate [programmatic modification of test data](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#programmatic-modification-of-test-data). |
| --prerebotmodifier <name:args> | Activate [programmatic modification of results](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#programmatic-modification-of-results). |
| --randomize <all|suites|tests|none> | [Randomizes](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#randomizing-execution-order) test execution order. |
| --console <verbose|dotted|quiet|none> | [Console output type](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#console-output-type). |
| --dotted | Shortcut for --console dotted. |
| --quiet | Shortcut for --console quiet. |
| -W, --consolewidth <width> | [Sets the width](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#console-width) of the console output. |
| -C, --consolecolors <auto|on|ansi|off> | [Specifies are colors](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#console-colors) used on the console. |
| --consolelinks <auto|off> | Controls [making paths to results files hyperlinks](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#console-links). |
| -K, --consolemarkers <auto|on|off> | Show [markers on the console](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#console-markers) when top level keywords in a test case end. |
| -P, --pythonpath <path> | Additional locations to add to the [module search path](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#module-search-path). |
| -A, --argumentfile <path> | A text file to [read more arguments](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#argument-files) from. |

## Test Case Failures

A test case **fails if any of the keyword it uses fails**. Normally this means that execution of that test case is stopped, possible test teardown is executed, and then execution continues from the next test case.

### Continuing Test Cases on Failures

**Way 1**: With the Run Keyword And Continue On Failure keyword on a test case or a user keyword, it's possible to keep running the test case to the end when stopping its execution is not desired. For example:

**\*\*\* Test Cases \*\*\***

Test case 1

Run Keyword and Continue on Failure Should be Equal 1 2

Log This is executed but test fails in the end

**\*\*\* Keywords \*\*\***

User keyword 1

Run Keyword and Continue on Failure Should be Equal 3 4

Log This is executed but test fails in the end

**Way 2**: Another way is to use the robot:continue-on-failure tag on a test case or a user keyword:

**\*\*\* Test Cases \*\*\***

Example

[Tags] robot:continue-on-failure

FOR ${index} IN RANGE 10 # Execute 10 times even when first time failed

Do Something

END

**\*\*\* Keywords \*\*\***

Example

[Tags] robot:continue-on-failure

Should be Equal 1 2

Log This is executed but test fails in the end

But setting robot:continue-on-failure within a test case or a user keyword will not propagate the continue-on-failure behavior into user keywords they call. If such recursive behavior is needed, the robot:recursive-continue-on-failure tag can be used. For example, all keywords in the following example are executed:

**\*\*\* Test Cases \*\*\***

Example

[Tags] robot:recursive-continue-on-failure

Should be Equal 1 2

User Keyword 1

Log This is executed

**\*\*\* Keywords \*\*\***

User Keyword 1

Should be Equal 3 4

Log This is executed

## Parallel Execution

<https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#parallel-execution-of-keywords>

## Dry Run

In *dry run* mode, tests are run normally otherwise, but the **keywords coming from the test libraries are not executed** at all. This mode is helpful to validate the test data. You can trigger it using --dryrun option .

The dry run execution may **fail** for following reasons:

* Using keywords that are **not found**.
* Using keywords with **wrong number of arguments**.
* Using user keywords that have **invalid syntax**.

The dry run execution may **show warnings** for following reasons:

* Test library or resource file imports cannot be resolved.
* Keyword is deprecated.

It is possible to **disable dry run validation of specific user keywords** by adding a special robot:no-dry-run [keyword tag](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#user-keyword-tags)to them.

**Notes:**

* The dry run mode does not validate variables.

# Debugging Tests

<https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#toc-entry-462>

## Log Levels

Messages in [log files](#_File_Types) can have different log levels. Some of the messages are written by Robot Framework itself, but also executed keywords can log information using different levels. The available log levels are:

|  |  |
| --- | --- |
| **Log level** | **Description** |
| FAIL | Used when a keyword fails. Can be used only by Robot Framework itself. |
| WARN | Used to display warnings. They shown also in [the console and in the Test Execution Errors section in log files](https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#errors-and-warnings-during-execution), but they do not affect the test case status. |
| INFO | The default level for normal messages. By default, messages below this level are not shown in the log file. |
| DEBUG | Used for debugging purposes. Useful, for example, for logging what libraries are doing internally. When a keyword fails, a traceback showing where in the code the failure occurred is logged using this level automatically. |
| TRACE | More detailed debugging level. The keyword arguments and return values are automatically logged using this level. |

By default, log messages below the INFO level are not logged, but this threshold can be changed from the command line using the --loglevel (-L) option. This option takes any of the available log levels as an argument, and that level becomes the new threshold level. A special value NONE can also be used to disable logging altogether.

Another way to change the log level is using the keyword Set Log Level in the test data.

# Listener Interface

Robot Framework's listener interface provides a **powerful mechanism for getting notifications and for inspecting and modifying data and results during execution**. Listeners are called, for example, when suites, tests and keywords start and end, when output files are ready, and finally when the whole execution ends. Example usages include **communicating with external test management systems**, sending a message when a test fails, and modifying tests during execution.

<https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#listener-interface>

# Remote Library Interface

<https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#remote-library-interface>

# Robot Best Practices

## Most Important Goals When Writing Test Cases

* Easy to understand
* Easy to maintain
* Fast to execute

Details:

<https://github.com/robotframework/HowToWriteGoodTestCases>

<https://www.slideshare.net/slideshow/robot-framework-dos-and-donts/38564910#4>

# Supported Tools

## Built-In Tools

<https://robotframework.org/robotframework/#built-in-tools>

|  |  |
| --- | --- |
| **Tool** | **Introduction** |
| Rebot | Tool for generating logs and reports based on XML outputs and for combining multiple outputs together. |
| Libdoc | Tool for generating keyword documentation for test libraries and resource files. |
| Testdoc | Generates high level HTML documentation based on Robot Framework test cases. |
| Tidy | Tool for cleaning up and changing format of Robot Framework test data files. |

## Robotidy

A tool for autoformatting Robot Framework code. It’s similar to Tidy, but better.

<https://robotidy.readthedocs.io/en/stable/index.html>

## VS Code Extension

### Language Server

[Robot Framework Language Server](https://marketplace.visualstudio.com/items?itemName=robocorp.robotframework-lsp) extension for VSCode.

### Code Format

By default, VSCode Robot Framework use its built-in formatter Tidy, but you can change to use **Robotidy** with two extra steps:

1. If you have not installed robotidy, then install it:

$ pip install robotframework-tidy

1. In VSCode settings.json, add:

"robot.codeFormatter": "robotidy"

## Jenkins

<https://plugins.jenkins.io/robot/>