Selenium2Library

Library version: 3.0.0b1
Library scope: global
Named arguments: supported

Introduction

NOTE: Selenium2Library has been renamed to SeleniumLibrary since version 3.0. Nowadays Selenium2Library is just a thin wrapper to SeleniumLibrary that eases with transitioning to the new project. See SeleniumLibrary and Selenium2Library project pages for more information.

Selenium2Library is a web testing library for Robot Framework.

This document is about using Selenium2Library. For information about installation, support, and more please visit the project page.

Selenium2Library uses the Selenium 2 (WebDriver) libraries internally to control a web browser. See http://seleniumhq.org/docs/03_webdriver.html for more information on Selenium 2 and WebDriver.

Selenium2Library runs tests in a real browser instance. It should work in most modern browsers and can be used with both Python and Jython interpreters.

Before running tests

Prior to running test cases using Selenium2Library, Selenium2Library must be imported into your Robot test suite (see *importing* section), and the *Open Browser* keyword must be used to open a browser to the desired location.

--- Note important change starting with Version 1.7.0 release ---

Locating or specifying elements

All keywords in Selenium2Library that need to find an element on the page take an argument, either a *locator* or now a *webelement*. *locator* is a string that describes how to locate an element using a syntax specifying different location strategies. *webelement* is a variable that holds a WebElement instance, which is a representation of the element.

Using locators

The locator can be used in two ways. In explicit way, where the strategy of the locator is defined as prefix in the locator or in implicit way, where there strategy is determined from the locator.

The implicit way supports two strategies: *xpath* and matching against *id* and *name* attributes. If locator starts with *//* or *(//* then *xpath* strategy will be used. Determining *(//* as xpath is supported from release 3.0.0 onwards. If locator does not start with *//* or *(//*, then it is matched against the *id* and *name* key attributes of elements. Example

```
Click Element    my_element    # id and name attribute matching
Click Element    //div         # xpath
Click Element    (//div)[2]    # xpath
```

In the explicit way, it is possible to specify the approach Selenium2Library should take to find an element by specifying a lookup strategy with a locator prefix. Supported strategies are:

Strategy	Example	Description
identifier	Click Element | identifier=my_element	Matches by @id or @name attribute
id	Click Element | id=my_element	Matches by @id attribute
name	Click Element | name=my_element	Matches by @name attribute
xpath	Click Element | xpath=//div[@id='my_element']	Matches with arbitrary XPath expression
dom	Click Element | dom=document.images[56]	Matches with arbitrary DOM express
link	Click Element | link=My Link	Matches anchor elements by their link text
partial link	Click Element | partial link=y Lin	Matches anchor elements by their partial link text
css	Click Element | css=div.my_class	Matches by CSS selector
class	Click Element | class=my_class	Matches by class name selector
jquery	Click Element | jquery=div.my_class	Matches by jQuery/sizzle selector
sizzle	Click Element | sizzle=div.my_class	Matches by jQuery/sizzle selector
tag	Click Element | tag=div	Matches by HTML tag name
default*	Click Link | default=page?a=b	Matches key attributes with value after first '='

* Explicitly specifying the default strategy is only necessary if locating elements by matching key attributes is desired and an attribute value contains a '='. The following would fail because it appears as if *page?a* is the specified lookup strategy:

```
Click Link    page?a=b
```

This can be fixed by changing the locator to:

```
Click Link    default=page?a=b
```

Please note that jQuery is not provided by Selenium2Library and if there is need to use jQuery locators, the system under test must provide the jQuery library.

Using webelements

Starting with version 1.7 of the Selenium2Library, one can pass an argument that contains a WebElement instead of a string locator. To get a WebElement, use the new *Get WebElements* keyword. For example:

${elem} =	Get WebElement	id=my_element
Click Element	${elem}	

Locating Tables, Table Rows, Columns, etc.

Table related keywords, such as *Table Should Contain*, work differently. By default, when a table locator value is provided, it will search for a table with the specified *id* attribute. For example:

```
Table Should Contain    my_table    text
```

More complex table lookup strategies are also supported:

Strategy	Example	Description
css	Table Should Contain | css=table.my_class | text	Matches by @id or @name attribute
xpath	Table Should Contain | xpath=//table/[@name="my_table"] | text	Matches by @id or @name attribute

Custom Locators

If more complex lookups are required than what is provided through the default locators, custom lookup strategies can be created. Using custom locators is a two part process. First, create a keyword that returns the WebElement that should be acted on.

Custom Locator Strategy	[Arguments]	${browser}	${criteria}		${tag}	${constraints}
	${retVal}=	Execute Javascript	return window.document.getElementById('${criteria}');			
	[Return]	${retVal}				

This keyword is a reimplementation of the basic functionality of the *id* locator where *${browser}* is a reference to the WebDriver instance and *${criteria}* is the text of the locator (i.e. everything that comes after the = sign). To use this locator it must first be registered with *Add Location Strategy*.

```
Add Location Strategy    custom    Custom Locator Strategy
```

The first argument of *Add Location Strategy* specifies the name of the lookup strategy (which must be unique). After registration of the lookup strategy, the usage is the same as other locators. See *Add Location Strategy* for more details.

Timeouts

There are several *Wait ...* keywords that take timeout as an argument. All of these timeout arguments are optional. The timeout used by all of them can be set globally using the *Set Selenium Timeout* keyword. The same timeout also applies to *Execute Async Javascript*.

All timeouts can be given as numbers considered seconds (e.g. 0.5 or 42) or in Robot Framework's time syntax (e.g. '1.5 seconds' or '1 min 30 s'). For more information about the time syntax see the *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 either empty or case-insensitively equal to `false`, `no` or `none`. Other strings are considered true regardless their value, and other argument types are tested using same *rules as in Python*.

True examples:

Set Screenshot Directory	${OUTPUT_DIR}/screenshots	persist=True	# Strings are generally true.
Set Screenshot Directory	${OUTPUT_DIR}/screenshots	persist=yes	# Same as the above.
Set Screenshot Directory	${OUTPUT_DIR}/screenshots	persist=${TRUE}	# Python True is true.
Set Screenshot Directory	${OUTPUT_DIR}/screenshots	persist=${42}	# Numbers other than 0 are true.

False examples:

Set Screenshot Directory	${OUTPUT_DIR}/screenshots	persist=False	# String false is false.
Set Screenshot Directory	${OUTPUT_DIR}/screenshots	persist=no	# Also string no is false.
Set Screenshot Directory	${OUTPUT_DIR}/screenshots	persist=NONE	# String NONE is false.
Set Screenshot Directory	${OUTPUT_DIR}/screenshots	persist=${EMPTY}	# Empty string is false.
Set Screenshot Directory	${OUTPUT_DIR}/screenshots	persist=${FALSE}	# Python False is false.
Set Screenshot Directory	${OUTPUT_DIR}/screenshots	persist=${NONE}	# Python None is false.

Note that prior to Selenium2Library 3.0, all non-empty strings, including `false`, `no` and `none`, were considered true.

Importing

Arguments	Documentation
timeout=5.0, *implicit_wait=0.0*, *run_on_failure=Capture Page Screenshot*, *screenshot_root_directory=None*	Selenium2Library can be imported with optional arguments. *timeout* is the default timeout used to wait for all waiting actions. It can be later set with *Set Selenium Timeout*. 'implicit_wait' is the implicit timeout that Selenium waits when looking for elements. It can be later set with *Set Selenium Implicit Wait*. See *WebDriver: Advanced Usage__* section of the SeleniumHQ documentation for more information about WebDriver's implicit wait functionality. __ http://seleniumhq.org/docs/04_webdriver_advanced.html#explicit-and-implicit-waits *run_on_failure* specifies the name of a keyword (from any available libraries) to execute when a Selenium2Library keyword fails. By default *Capture Page Screenshot* will be used to take a screenshot of the current page. Using the value "Nothing" will disable this feature altogether. See *Register Keyword To Run On Failure* keyword for more information about this functionality. *screenshot_root_directory* specifies the default root directory that screenshots should be stored in. If not provided the default directory will be where robotframework places its logfile. Examples: <table><tr><td>Library | Selenium2Library | 15</td><td># Sets default timeout to 15 seconds</td></tr><tr><td>Library | Selenium2Library | 0 | 5</td><td># Sets default timeout to 0 seconds and default implicit_wait to 5 seconds</td></tr><tr><td>Library | Selenium2Library | 5 | run_on_failure=Log Source</td><td># Sets default timeout to 5 seconds and runs *Log Source* on failure</td></tr><tr><td>Library | Selenium2Library | implicit_wait=5 | run_on_failure=Log Source</td><td># Sets default implicit_wait to 5 seconds and runs *Log Source* on failure</td></tr><tr><td>Library | Selenium2Library | timeout=10 | run_on_failure=Nothing</td><td># Sets default timeout to 10 seconds and does nothing on failure</td></tr></table>

Shortcuts

Add Cookie · Add Location Strategy · Alert Should Be Present · Assign Id To Element · Capture Page Screenshot · Checkbox Should Be Selected · Checkbox Should Not Be Selected · Choose Cancel On Next Confirmation · Choose File · Choose Ok On Next Confirmation · Clear Element Text · Click Button · Click Element · Click Element At Coordinates · Click Image · Click Link · Close All Browsers · Close Browser · Close Window · Confirm Action · Create Webdriver · Current Frame Contains · Current Frame Should Not Contain · Delete All Cookies · Delete Cookie · Dismiss Alert · Double Click Element · Drag And Drop · Drag And Drop By Offset · Element Should Be Disabled · Element Should Be Enabled · Element Should Be Focused · Element Should Be Visible · Element Should Contain · Element Should Not Be Visible · Element Should Not Contain · Element Text Should Be · Execute Async Javascript · Execute Javascript · Focus · Frame Should Contain · Get Alert Message · Get All Links · Get Cookie Value · Get Cookies · Get Element Attribute · Get Element Size · Get Horizontal Position · Get List Items · Get Location · Get Locations · Get Log · Get Matching Xpath Count · Get Selected List Label · Get Selected List Labels · Get Selected List Value · Get Selected List Values · Get Selenium Implicit Wait · Get Selenium Speed · Get Selenium Timeout · Get Source · Get Table Cell · Get Text · Get Title · Get Value · Get Vertical Position · Get Webelement · Get Webelements · Get Window Identifiers · Get Window Names · Get Window Position · Get Window Size · Get Window Titles · Go Back · Go To · Input Password · Input Text · Input Text Into Prompt · List Selection Should Be · List Should Have No Selections · List Windows · Location Should Be · Location Should Contain · Locator Should Match X Times · Log Location · Log Source · Log Title · Maximize Browser Window · Mouse Down · Mouse Down On Image · Mouse Down On Link · Mouse Out · Mouse Over · Mouse Up · Open Browser · Open Context Menu · Page Should Contain · Page Should Contain Button · Page Should Contain Checkbox · Page Should Contain Element · Page Should Contain Image · Page Should Contain Link · Page Should Contain List · Page Should Contain Radio Button · Page Should Contain Textfield · Page Should Not Contain · Page Should Not Contain Button · Page Should Not Contain Checkbox · Page Should Not Contain Element · Page Should Not Contain Image · Page Should Not Contain Link · Page Should Not Contain List · Page Should Not Contain Radio Button · Page Should Not Contain Textfield · Press Key · Radio Button Should Be Set To · Radio Button Should Not Be Selected · Register Keyword To Run On Failure · Reload Page · Remove Location Strategy · Select All From List · Select Checkbox · Select Frame · Select From List · Select From List By Index · Select From List By Label · Select From List By Value · Select Radio Button · Select Window · Set Browser Implicit Wait · Set Screenshot Directory · Set Selenium Implicit Wait · Set Selenium Speed · Set Selenium Timeout · Set Window Position · Set Window Size · Simulate · Submit Form · Switch Browser · Table Cell Should Contain · Table Column Should Contain · Table Footer Should Contain · Table Header Should Contain · Table Row Should Contain · Textarea Should Contain · Textarea Value Should Be · Textfield Should Contain · Textfield Value Should Be · Title Should Be · Unselect Checkbox · Unselect Frame · Unselect From List · Unselect From List By Index · Unselect From List By Label · Unselect From List By Value · Wait For Condition · Wait Until Element Contains · Wait Until Element Does Not Contain · Wait Until Element Is Enabled · Wait Until Element Is Not Visible · Wait Until Element Is Visible · Wait Until Page Contains · Wait Until Page Contains Element · Wait Until Page Does Not Contain · Wait Until Page Does Not Contain Element · Xpath Should Match X Times

Keywords

Keyword	Arguments	Documentation
Add Cookie	*name*, *value*, *path=None*, *domain=None*, *secure=None*	Adds a cookie to your current session. "name" and "value" are required, "path", "domain" and "secure" are optional
Add Location Strategy	*strategy_name*, *strategy_keyword*, *persist=False*	Adds a custom location strategy based on a keyword. Location strategies are automatically removed after leaving the current scope by default. Setting *persist* to Python True will cause the location strategy to stay registered throughout the life of the test. Trying to add a custom location strategy with the same name as one that already exists will cause the keyword to fail. Custom locator keyword example: <table><tr><td>Custom Locator Strategy</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td>[Arguments]</td><td>${browser}</td><td>${criteria}</td><td>${tag}</td><td>${constraints}</td></tr><tr><td></td><td>${retVal}=</td><td>Execute Javascript</td><td>return window.document.getElementById('${criteria}');</td><td></td><td></td></tr><tr><td></td><td>[Return]</td><td>${retVal}</td><td></td><td></td><td></td></tr></table> Usage example: <table><tr><td>Add Location Strategy</td><td>custom</td><td>Custom Locator Strategy</td></tr><tr><td>Page Should Contain Element</td><td>custom=my_id</td><td></td></tr></table> See *Remove Location Strategy* for details about removing a custom location strategy.
Alert Should Be Present	*text=*	Verifies an alert is present and dismisses it. If *text* is a non-empty string, then it is also verified that the message of the alert equals to *text*. Will fail if no alert is present. Note that following keywords will fail unless the alert is dismissed by this keyword or another like *Get Alert Message*.
Assign Id To Element	*locator*, *id*	Assigns a temporary identifier to element specified by *locator*. This is mainly useful if the locator is complicated/slow XPath expression. Identifier expires when the page is reloaded. Example: <table><tr><td>Assign ID to Element</td><td>xpath=//div[@id="first_div"]</td><td>my id</td></tr><tr><td>Page Should Contain Element</td><td>my id</td><td></td></tr></table>
Capture Page Screenshot	*filename=selenium-screenshot-{index}.png*	Takes a screenshot of the current page and embeds it into the log. `filename` argument specifies the name of the file to write the screenshot into. If no `filename` is given, the screenshot is saved into file *selenium-screenshot-{index}.png* under the directory where the Robot Framework log file is written into. The `filename` is also considered relative to the same directory, if it is not given in absolute format. If an absolute or relative path is given but the path does not exist it will be created. Starting from Selenium2Library 1.8 if `filename` contains *{index}* characters, it will be automatically replaced with running index. The running index is unique for each different filename. The absolute path of the saved screenshot is always returned and it does not depend does the `filename` contain *{index}*. See example 1 and 2 for more details. The *{index}* is replaced with the actual index by using Python's str.format method, and it can be formatted using the standard format string syntax. The example 3 shows this by setting the width and the fill character. If there is a need to write literal *{index}* or if `filename` contains *{* or *}* characters, then the braces must be doubled. If *{index}* is used, the computed filename will not overwrite an existing file. The number chosen will be the first number that results in a unique filename. For example, if the computed name is screenshot-1.png but screenshot-1.png already exists, screenshot-2.png will be tried, and so on, until a unique name is found. Example 1: <table><tr><td>${file1} =</td><td>Capture Page Screenshot</td><td></td></tr><tr><td>File Should Exist</td><td>${OUTPUTDIR}${/}selenium-screenshot-1.png</td><td></td></tr><tr><td>Should Be Equal</td><td>${file1}</td><td>${OUTPUTDIR}${/}selenium-screenshot-1.png</td></tr><tr><td>${file2} =</td><td>Capture Page Screenshot</td><td></td></tr><tr><td>File Should Exist</td><td>${OUTPUTDIR}${/}selenium-screenshot-2.png</td><td></td></tr><tr><td>Should Be Equal</td><td>${file2}</td><td>${OUTPUTDIR}${/}selenium-screenshot-2.png</td></tr></table> Example 2: <table><tr><td>${file1} =</td><td>Capture Page Screenshot</td><td>${OTHER_DIR}${/}other-{index}-name.png</td></tr><tr><td>${file2} =</td><td>Capture Page Screenshot</td><td>${OTHER_DIR}${/}some-other-name-{index}.png</td></tr><tr><td>${file3} =</td><td>Capture Page Screenshot</td><td>${OTHER_DIR}${/}other-{index}-name.png</td></tr><tr><td>File Should Exist</td><td>${OTHER_DIR}${/}other-1-name.png</td><td></td></tr><tr><td>Should Be Equal</td><td>${file1}</td><td>${OTHER_DIR}${/}other-1-name.png</td></tr><tr><td>File Should Exist</td><td>${OTHER_DIR}${/}some-other-name-1.png</td><td></td></tr><tr><td>Should Be Equal</td><td>${file2}</td><td>${OTHER_DIR}${/}some-other-name-1.png</td></tr><tr><td>File Should Exist</td><td>${OTHER_DIR}${/}other-2-name.png</td><td></td></tr><tr><td>Should Be Equal</td><td>${file3}</td><td>${OTHER_DIR}${/}other-2-name.png</td></tr></table> Example 3: <table><tr><td>Capture Page Screenshot</td><td>${OTHER_DIR}${/}sc-{index:06}.png</td></tr><tr><td>File Should Exist</td><td>${OTHER_DIR}${/}sc-000001.png</td></tr></table>
Checkbox Should Be Selected	*locator*	Verifies checkbox identified by *locator* is selected/checked. Key attributes for checkboxes are *id* and *name*. See *introduction* for details about locating elements.
Checkbox Should Not Be Selected	*locator*	Verifies checkbox identified by *locator* is not selected/checked. Key attributes for checkboxes are *id* and *name*. See *introduction* for details about locating elements.
Choose Cancel On Next Confirmation		Cancel will be selected the next time *Confirm Action* is used.
Choose File	*locator*, *file_path*	Inputs the *file_path* into file input field found by *locator*. This keyword is most often used to input files into upload forms. The file specified with *file_path* must be available on the same host where the Selenium Server is running. Example: <table><tr><td>Choose File</td><td>my_upload_field</td><td>/home/user/files/trades.csv</td></tr></table>

Choose Ok On Next Confirmation		Undo the effect of using keywords *Choose Cancel On Next Confirmation*. Note that Selenium's overridden window.confirm() function will normally automatically return true, as if the user had manually clicked OK, so you shouldn't need to use this command unless for some reason you need to change your mind prior to the next confirmation. After any confirmation, Selenium will resume using the default behavior for future confirmations, automatically returning true (OK) unless/until you explicitly use *Choose Cancel On Next Confirmation* for each confirmation.
		Note that every time a confirmation comes up, you must consume it by using a keywords such as *Get Alert Message*, or else the following selenium operations will fail.
Clear Element Text	*locator*	Clears the text value of text entry element identified by *locator*.
		See *introduction* for details about locating elements.
Click Button	*locator*	Clicks a button identified by *locator*.
		Key attributes for buttons are *id*, *name* and *value*. See *introduction* for details about locating elements.
Click Element	*locator*	Click element identified by *locator*.
		Key attributes for arbitrary elements are *id* and *name*. See *introduction* for details about locating elements.
Click Element At Coordinates	*locator*, *xoffset*, *yoffset*	Click element identified by *locator* at x/y coordinates of the element. Cursor is moved and the center of the element and x/y coordinates are calculted from that point.
		Key attributes for arbitrary elements are *id* and *name*. See *introduction* for details about locating elements.
Click Image	*locator*	Clicks an image found by *locator*.
		Key attributes for images are *id*, *src* and *alt*. See *introduction* for details about locating elements.
Click Link	*locator*	Clicks a link identified by locator.
		Key attributes for links are *id*, *name*, *href* and link text. See *introduction* for details about locating elements.
Close All Browsers		Closes all open browsers and resets the browser cache.
		After this keyword new indexes returned from *Open Browser* keyword are reset to 1.
		This keyword should be used in test or suite teardown to make sure all browsers are closed.
Close Browser		Closes the current browser.
Close Window		Closes currently opened pop-up window.
Confirm Action		Dismisses currently shown confirmation dialog and returns it's message.
		By default, this keyword chooses 'OK' option from the dialog. If 'Cancel' needs to be chosen, keyword *Choose Cancel On Next Confirmation* must be called before the action that causes the confirmation dialog to be shown.
		Examples:

Click Button	Send	# Shows a confirmation dialog
${message}=	Confirm Action	# Chooses Ok
Should Be Equal	${message}	Are your sure?
Choose Cancel On Next Confirmation		
Click Button	Send	# Shows a confirmation dialog
Confirm Action		# Chooses Cancel

Create Webdriver	*driver_name*, *alias=None*, *kwargs={}*, ***init_kwargs*	Creates an instance of a WebDriver.
		Like *Open Browser*, but allows passing arguments to a WebDriver's __init__. *Open Browser* is preferred over *Create Webdriver* when feasible.
		Returns the index of this browser instance which can be used later to switch back to it. Index starts from 1 and is reset back to it when *Close All Browsers* keyword is used. See *Switch Browser* for example.
		driver_name must be the exact name of a WebDriver in *selenium.webdriver* to use. WebDriver names include: Firefox, Chrome, Ie, Opera, Safari, PhantomJS, and Remote.
		Use keyword arguments to specify the arguments you want to pass to the WebDriver's __init__. The values of the arguments are not processed in any way before being passed on. For Robot Framework < 2.8, which does not support keyword arguments, create a keyword dictionary and pass it in as argument *kwargs*. See the *Selenium API Documentation* for information about argument names and appropriate argument values.
		Examples:

# use proxy for Firefox			
${proxy}=	Evaluate	sys.modules['selenium.webdriver'].Proxy()	sys, selenium.webdriver
${proxy.http_proxy}=	Set Variable	localhost:8888	
Create Webdriver	Firefox	proxy=${proxy}	
# use a proxy for PhantomJS			
${service args}=	Create List	--proxy=192.168.132.104:8888	
Create Webdriver	PhantomJS	service_args=${service args}	

Example for Robot Framework < 2.8:

# debug IE driver			
${kwargs}=	Create Dictionary	log_level=DEBUG	log_file=%{HOMEPATH}${/}ie.log
Create Webdriver	Ie	kwargs=${kwargs}	

Current Frame Contains	*text*, *loglevel=INFO*	Verifies that current frame contains *text*.
		See *Page Should Contain* for explanation about *loglevel* argument.
Current Frame Should Not Contain	*text*, *loglevel=INFO*	Verifies that current frame contains *text*.
		See *Page Should Contain* for explanation about *loglevel* argument.
Delete All Cookies		Deletes all cookies.
Delete Cookie	*name*	Deletes cookie matching *name*.
		If the cookie is not found, nothing happens.
Dismiss Alert	*accept=True*	Returns true if alert was confirmed, false if it was dismissed
		This keyword will fail if no alert is present. Note that following keywords will fail unless the alert is dismissed by this keyword or another like *Get Alert Message*.

Double Click Element	*locator*	Double click element identified by *locator*.
		Key attributes for arbitrary elements are *id* and *name*. See *introduction* for details about locating elements.
Drag And Drop	*source*, *target*	Drags element identified with *source* which is a locator.
		Element can be moved on top of another element with *target* argument.
		target is a locator of the element where the dragged object is dropped.
		Examples:
		Drag And Drop | elem1 | elem2 | # Move elem1 over elem2.
Drag And Drop By Offset	*source*, *xoffset*, *yoffset*	Drags element identified with *source* which is a locator.
		Element will be moved by xoffset and yoffset, each of which is a negative or positive number specify the offset.
		Examples:
		Drag And Drop By Offset | myElem | 50 | -35 | # Move myElem 50px right and 35px down.
Element Should Be Disabled	*locator*	Verifies that element identified with *locator* is disabled.
		Key attributes for arbitrary elements are *id* and *name*. See *introduction* for details about locating elements.
Element Should Be Enabled	*locator*	Verifies that element identified with *locator* is enabled.
		Key attributes for arbitrary elements are *id* and *name*. See *introduction* for details about locating elements.
Element Should Be Focused	*locator*	Verifies that element identified with *locator* is focused.
		Key attributes for arbitrary elements are *id* and *name*. See *introduction* for details about locating elements.
		New in Selenium2Library 3.0.0.
Element Should Be Visible	*locator*, *message=*	Verifies that the element identified by *locator* is visible.
		Herein, visible means that the element is logically visible, not optically visible in the current browser viewport. For example, an element that carries display:none is not logically visible, so using this keyword on that element would fail.
		message can be used to override the default error message.
		Key attributes for arbitrary elements are *id* and *name*. See *introduction* for details about locating elements.
Element Should Contain	*locator*, *expected*, *message=*	Verifies element identified by *locator* contains text *expected*.
		If you wish to assert an exact (not a substring) match on the text of the element, use *Element Text Should Be*.
		message can be used to override the default error message.
		Key attributes for arbitrary elements are *id* and *name*. See *introduction* for details about locating elements.
Element Should Not Be Visible	*locator*, *message=*	Verifies that the element identified by *locator* is NOT visible.
		This is the opposite of *Element Should Be Visible*.
		message can be used to override the default error message.
		Key attributes for arbitrary elements are *id* and *name*. See *introduction* for details about locating elements.
Element Should Not Contain	*locator*, *expected*, *message=*	Verifies element identified by *locator* does not contain text *expected*.
		message can be used to override the default error message.
		Key attributes for arbitrary elements are *id* and *name*. See *Element Should Contain* for more details.
Element Text Should Be	*locator*, *expected*, *message=*	Verifies element identified by *locator* exactly contains text *expected*.
		In contrast to *Element Should Contain*, this keyword does not try a substring match but an exact match on the element identified by *locator*.
		message can be used to override the default error message.
		Key attributes for arbitrary elements are *id* and *name*. See *introduction* for details about locating elements.
Execute Async Javascript	**code*	Executes asynchronous JavaScript code.
		Similar to *Execute Javascript* except that scripts executed with this keyword must explicitly signal they are finished by invoking the provided callback. This callback is always injected into the executed function as the last argument.
		Scripts must complete within the script timeout or this keyword will fail. See the *Timeouts* section for more information.
		Examples:
Execute Javascript	**code*	Executes the given JavaScript code.
		code may contain multiple lines of code and may be divided into multiple cells in the test data. In that case, the parts are catenated together without adding spaces.
		If *code* is an absolute path to an existing file, the JavaScript to execute will be read from that file. Forward slashes work as a path separator on all operating systems.
		The JavaScript executes in the context of the currently selected frame or window as the body of an anonymous function. Use *window* to refer to the window of your application and *document* to refer to the document object of the current frame or window, e.g. *document.getElementById('foo')*.
		This keyword returns None unless there is a return statement in the JavaScript. Return values are converted to the appropriate type in Python, including WebElements.
		Examples:

Execute Async Javascript examples:

Execute Async JavaScript	var callback = arguments[arguments.length - 1];	window.setTimeout(callback, 2000);
Execute Async JavaScript	${CURDIR}/async_js_to_execute.js	
${retval}=	Execute Async JavaScript	
...	var callback = arguments[arguments.length - 1];	
...	function answer(){callback("text");};	
...	window.setTimeout(answer, 2000);	
Should Be Equal	${retval}	text

Execute Javascript examples:

Execute JavaScript	window.my_js_function('arg1', 'arg2')	
Execute JavaScript	${CURDIR}/js_to_execute.js	
${sum}=	Execute JavaScript	return 1 + 1;
Should Be Equal	${sum}	${2}

Focus	*locator*	Sets focus to element identified by *locator*.
Frame Should Contain	*locator*, *text*, *loglevel=INFO*	Verifies frame identified by *locator* contains *text*.
		See *Page Should Contain* for explanation about *loglevel* argument.
		Key attributes for frames are *id* and *name*. See *introduction* for details about locating elements.
Get Alert Message	*dismiss=True*	Returns the text of current JavaScript alert.
		By default the current JavaScript alert will be dismissed. This keyword will fail if no alert is present. Note that following keywords will fail unless the alert is dismissed by this keyword or another like *Dismiss Alert*.
Get All Links		Returns a list containing ids of all links found in current page.
		If a link has no id, an empty string will be in the list instead.
Get Cookie Value	*name*	Returns value of cookie found with *name*.
		If no cookie is found with *name*, this keyword fails.
Get Cookies		Returns all cookies of the current page.
Get Element Attribute	*locator*, *attribute_name=None*	Returns value of the element attribute.
		There are two cases how to use this keyword.
		First, if only *locator* is provided, *locator* should consists of element locator followed by an @ sign and attribute name. This behavior is left for backward compatibility.
		Example:
		${id}= Get Element Attribute link=Link with id@id
		Second, if *locator* and *attribute_name* are provided both, *locator* should be standard locator and *attribute_name* is name of the requested element attribute.
		Examples:
		${id}= Get Element Attribute link=Link with id id ${element_by_dom}= Get Webelement dom=document.getElementsByTagName('a')[3] ${id}= Get Element Attribute ${element_by_dom} id
Get Element Size	*locator*	Returns width and height of element identified by *locator*.
		The element width and height is returned. Fails if a matching element is not found.
Get Horizontal Position	*locator*	Returns horizontal position of element identified by *locator*.
		The position is returned in pixels off the left side of the page, as an integer. Fails if a matching element is not found.
		See also *Get Vertical Position*.
Get List Items	*locator*, *value=False*	Returns the labels or values in the select list identified by *locator*.
		Select list keywords work on both lists and combo boxes. Key attributes for select lists are *id* and *name*. See *introduction* for details about locating elements.
		Example:
		${labels1} = Get List Items xpath=//h1 ${labels2} = Get List Items xpath=//h1 value=${False} ${values} = Get List Items xpath=//h1 value=True Should Be Equal ${labels1} ${labels2}
Get Location		Returns the current location.
Get Locations		Returns and logs current locations of all windows known to the browser.
Get Log	*log_type*	Get the log for a given selenium log type
		The *log_type* argument defines which logs to get. Possible values are: *browser*, *driver*, *client* or *server*
		New in Selenium2Library 3.0
Get Matching Xpath Count	*xpath*, *return_str=True*	Returns number of elements matching *xpath*
		The default return type is *str* but it can changed to *int* by setting the `return_str` argument to Python False.
		One should not use the xpath= prefix for 'xpath'. XPath is assumed.
		Correct:
| | | ```
count = | Get Matching Xpath Count | //div[@id='sales-pop']
``` |
| | | Incorrect: |
| | | ```
count = | Get Matching Xpath Count | xpath=//div[@id='sales-pop']
``` |
		If you wish to assert the number of matching elements, use *Xpath Should Match X Times*.
Get Selected List Label	*locator*	Returns the visible label of the selected element from the select list identified by *locator*.
		Select list keywords work on both lists and combo boxes. Key attributes for select lists are *id* and *name*. See *introduction* for details about locating elements.
Get Selected List Labels	*locator*	Returns the visible labels of selected elements (as a list) from the select list identified by *locator*.
		Fails if there is no selection.
		Select list keywords work on both lists and combo boxes. Key attributes for select lists are *id* and *name*. See *introduction* for details about locating elements.
Get Selected List Value	*locator*	Returns the value of the selected element from the select list identified by *locator*.
		Return value is read from *value* attribute of the selected element.
		Select list keywords work on both lists and combo boxes. Key attributes for select lists are *id* and *name*. See *introduction* for details about locating elements.
Get Selected List Values	*locator*	Returns the values of selected elements (as a list) from the select list identified by *locator*.
		Fails if there is no selection.

		Select list keywords work on both lists and combo boxes. Key attributes for select lists are *id* and *name*. See *introduction* for details about locating elements.
Get Selenium Implicit Wait		Gets the wait in seconds that is waited by Selenium. See *Set Selenium Implicit Wait* for an explanation.
Get Selenium Speed		Gets the delay in seconds that is waited after each Selenium command. See *Set Selenium Speed* for an explanation.
Get Selenium Timeout		Gets the timeout in seconds that is used by various keywords. See *Set Selenium Timeout* for an explanation.
Get Source		Returns the entire html source of the current page or frame.
Get Table Cell	*table_locator*, *row*, *column*, *loglevel=INFO*	Returns the content from a table cell. Row and column number start from 1. Header and footer rows are included in the count. A negative row or column number can be used to get rows counting from the end (end: -1). Cell content from header or footer rows can be obtained with this keyword. To understand how tables are identified, please take a look at the *introduction*. See *Page Should Contain* for explanation about *loglevel* argument.
Get Text	*locator*	Returns the text value of element identified by *locator*. See *introduction* for details about locating elements.
Get Title		Returns title of current page.
Get Value	*locator*	Returns the value attribute of element identified by *locator*. See *introduction* for details about locating elements.
Get Vertical Position	*locator*	Returns vertical position of element identified by *locator*. The position is returned in pixels off the top of the page, as an integer. Fails if a matching element is not found. See also *Get Horizontal Position*.
Get Webelement	*locator*	Returns the first WebElement matching the given locator. See *introduction* for details about locating elements.
Get Webelements	*locator*	Returns list of WebElement objects matching locator. See *introduction* for details about locating elements.
Get Window Identifiers		Returns and logs id attributes of all windows known to the browser.
Get Window Names		Returns and logs names of all windows known to the browser.
Get Window Position		Returns current window position as *x* then *y* (relative to the left and top of the screen). Example: `${x}` `${y}=` `Get Window Position`
Get Window Size		Returns current window size as *width* then *height*. Example: `${width}` `${height}=` `Get Window Size`
Get Window Titles		Returns and logs titles of all windows known to the browser.
Go Back		Simulates the user clicking the "back" button on their browser.
Go To	*url*	Navigates the active browser instance to the provided URL.
Input Password	*locator*, *text*	Types the given password into text field identified by *locator*. Difference between this keyword and *Input Text* is that this keyword does not log the given password. See *introduction* for details about locating elements.
Input Text	*locator*, *text*	Types the given *text* into text field identified by *locator*. See *introduction* for details about locating elements.
Input Text Into Prompt	*text*	Types the given *text* into alert box.
List Selection Should Be	*locator*, **items*	Verifies the selection of select list identified by *locator* is exactly **items*. If you want to test that no option is selected, simply give no *items*. Select list keywords work on both lists and combo boxes. Key attributes for select lists are *id* and *name*. See *introduction* for details about locating elements.
List Should Have No Selections	*locator*	Verifies select list identified by *locator* has no selections. Select list keywords work on both lists and combo boxes. Key attributes for select lists are *id* and *name*. See *introduction* for details about locating elements.
List Windows		Return all current window handles as a list
Location Should Be	*url*	Verifies that current URL is exactly *url*.
Location Should Contain	*expected*	Verifies that current URL contains *expected*.
Locator Should Match X Times	*locator*, *expected_locator_count*, *message=*, *loglevel=INFO*	Verifies that the page contains the given number of elements located by the given *locator*. See *introduction* for details about locating elements. See *Page Should Contain Element* for explanation about *message* and *loglevel* arguments.
Log Location		Logs and returns the current location.
Log Source	*loglevel=INFO*	Logs and returns the entire html source of the current page or frame. The *loglevel* argument defines the used log level. Valid log levels are WARN, INFO (default), DEBUG, and NONE (no logging).
Log Title		Logs and returns the title of current page.

Maximize Browser Window		Maximizes current browser window.
Mouse Down	*locator*	Simulates pressing the left mouse button on the element specified by *locator*.
		The element is pressed without releasing the mouse button.
		Key attributes for arbitrary elements are *id* and *name*. See *introduction* for details about locating elements.
		See also the more specific keywords *Mouse Down On Image* and *Mouse Down On Link*.
Mouse Down On Image	*locator*	Simulates a mouse down event on an image.
		Key attributes for images are *id*, *src* and *alt*. See *introduction* for details about locating elements.
Mouse Down On Link	*locator*	Simulates a mouse down event on a link.
		Key attributes for links are *id*, *name*, *href* and link text. See *introduction* for details about locating elements.
Mouse Out	*locator*	Simulates moving mouse away from the element specified by *locator*.
		Key attributes for arbitrary elements are *id* and *name*. See *introduction* for details about locating elements.
Mouse Over	*locator*	Simulates hovering mouse over the element specified by *locator*.
		Key attributes for arbitrary elements are *id* and *name*. See *introduction* for details about locating elements.
Mouse Up	*locator*	Simulates releasing the left mouse button on the element specified by *locator*.
		Key attributes for arbitrary elements are *id* and *name*. See *introduction* for details about locating elements.
Open Browser	*url*, *browser=firefox*, *alias=None*, *remote_url=False*, *desired_capabilities=None*, *ff_profile_dir=None*	Opens a new browser instance to given URL.

Returns the index of this browser instance which can be used later to switch back to it. Index starts from 1 and is reset back to it when *Close All Browsers* keyword is used. See *Switch Browser* for example.

Optional alias is an alias for the browser instance and it can be used for switching between browsers (just as index can be used). See *Switch Browser* for more details.

Possible values for *browser* are as follows:

firefox	FireFox
ff	FireFox
internetexplorer	Internet Explorer
ie	Internet Explorer
googlechrome	Google Chrome
gc	Google Chrome
chrome	Google Chrome
opera	Opera
phantomjs	PhantomJS
htmlunit	HTMLUnit
htmlunitwithjs	HTMLUnit with Javascipt support
android	Android
iphone	Iphone
safari	Safari
edge	Edge

Note, that you will encounter strange behavior, if you open multiple Internet Explorer browser instances. That is also why *Switch Browser* only works with one IE browser at most. For more information see: http://selenium-grid.seleniumhq.org/faq.html#i_get_some_strange_errors_when_i_run_multiple_internet_explorer_instances_on_the_same_machine

Optional 'remote_url' is the url for a remote selenium server for example http://127.0.0.1:4444/wd/hub. If you specify a value for remote you can also specify 'desired_capabilities' which is a string in the form key1:val1,key2:val2 that will be used to specify desired_capabilities to the remote server. This is useful for doing things like specify a proxy server for internet explorer or for specify browser and os if your using saucelabs.com. 'desired_capabilities' can also be a dictonary (created with 'Create Dictionary') to allow for more complex configurations.

Optional 'ff_profile_dir' is the path to the firefox profile dir if you wish to overwrite the default.

Open Context Menu	*locator*	Opens context menu on element identified by *locator*.
Page Should Contain	*text*, *loglevel=INFO*	Verifies that current page contains *text*.
		If this keyword fails, it automatically logs the page source using the log level specified with the optional *loglevel* argument. Valid log levels are DEBUG, INFO (default), WARN, and NONE. If the log level is NONE or below the current active log level the source will not be logged.
Page Should Contain Button	*locator*, *message=*, *loglevel=INFO*	Verifies button identified by *locator* is found from current page.
		This keyword searches for buttons created with either *input* or *button* tag.
		See *Page Should Contain Element* for explanation about *message* and *loglevel* arguments.
		Key attributes for buttons are *id*, *name* and *value*. See *introduction* for details about locating elements.
Page Should Contain Checkbox	*locator*, *message=*, *loglevel=INFO*	Verifies checkbox identified by *locator* is found from current page.
		See *Page Should Contain Element* for explanation about *message* and *loglevel* arguments.
		Key attributes for checkboxes are *id* and *name*. See *introduction* for details about locating elements.
Page Should Contain Element	*locator*, *message=*, *loglevel=INFO*	Verifies element identified by *locator* is found on the current page.
		message can be used to override default error message.
		See *Page Should Contain* for explanation about *loglevel* argument.
		Key attributes for arbitrary elements are *id* and *name*. See *introduction* for details about locating elements.
Page Should Contain Image	*locator*, *message=*, *loglevel=INFO*	Verifies image identified by *locator* is found from current page. See *Page Should Contain Element* for explanation about *message* and *loglevel* arguments.
		Key attributes for images are *id*, *src* and *alt*. See *introduction* for details about locating elements.
Page Should Contain Link	*locator*, *message=*, *loglevel=INFO*	Verifies link identified by *locator* is found from current page.
		See *Page Should Contain Element* for explanation about *message* and *loglevel* arguments.
		Key attributes for links are *id*, *name*, *href* and link text. See *introduction* for details about locating elements.

Page Should Contain List	locator, message=, loglevel=INFO	Verifies select list identified by locator is found from current page.
		See Page Should Contain Element for explanation about message and loglevel arguments.
		Key attributes for lists are id and name. See introduction for details about locating elements.
Page Should Contain Radio Button	locator, message=, loglevel=INFO	Verifies radio button identified by locator is found from current page.
		See Page Should Contain Element for explanation about message and loglevel arguments.
		Key attributes for radio buttons are id, name and value. See introduction for details about locating elements.
Page Should Contain Textfield	locator, message=, loglevel=INFO	Verifies text field identified by locator is found from current page.
		See Page Should Contain Element for explanation about message and loglevel arguments.
		Key attributes for text fields are id and name. See introduction for details about locating elements.
Page Should Not Contain	text, loglevel=INFO	Verifies the current page does not contain text.
		See Page Should Contain for explanation about loglevel argument.
Page Should Not Contain Button	locator, message=, loglevel=INFO	Verifies button identified by locator is not found from current page.
		This keyword searches for buttons created with either input or button tag.
		See Page Should Contain Element for explanation about message and loglevel arguments.
		Key attributes for buttons are id, name and value. See introduction for details about locating elements.
Page Should Not Contain Checkbox	locator, message=, loglevel=INFO	Verifies checkbox identified by locator is not found from current page.
		See Page Should Contain Element for explanation about message and loglevel arguments.
		Key attributes for checkboxes are id and name. See introduction for details about locating elements.
Page Should Not Contain Element	locator, message=, loglevel=INFO	Verifies element identified by locator is not found on the current page.
		message can be used to override the default error message.
		See Page Should Contain for explanation about loglevel argument.
		Key attributes for arbitrary elements are id and name. See introduction for details about locating elements.
Page Should Not Contain Image	locator, message=, loglevel=INFO	Verifies image identified by locator is found from current page.
		See Page Should Contain Element for explanation about message and loglevel arguments.
		Key attributes for images are id, src and alt. See introduction for details about locating elements.
Page Should Not Contain Link	locator, message=, loglevel=INFO	Verifies image identified by locator is found from current page.
		See Page Should Contain Element for explanation about message and loglevel arguments.
		Key attributes for images are id, src and alt. See introduction for details about locating elements.
Page Should Not Contain List	locator, message=, loglevel=INFO	Verifies select list identified by locator is not found from current page.
		See Page Should Contain Element for explanation about message and loglevel arguments.
		Key attributes for lists are id and name. See introduction for details about locating elements.
Page Should Not Contain Radio Button	locator, message=, loglevel=INFO	Verifies radio button identified by locator is not found from current page.
		See Page Should Contain Element for explanation about message and loglevel arguments.
		Key attributes for radio buttons are id, name and value. See introduction for details about locating elements.
Page Should Not Contain Textfield	locator, message=, loglevel=INFO	Verifies text field identified by locator is not found from current page.
		See Page Should Contain Element for explanation about message and loglevel arguments.
		Key attributes for text fields are id and name. See introduction for details about locating elements.
Press Key	locator, key	Simulates user pressing key on element identified by locator. key is either a single character, a string, or a numerical ASCII code of the key lead by '\\'. Examples:

Press Key	text_field	q	
Press Key	text_field	abcde	
Press Key	login_button	\\13	# ASCII code for enter key

Radio Button Should Be Set To	group_name, value	Verifies radio button group identified by group_name has its selection set to value.
		See Select Radio Button for information about how radio buttons are located.
Radio Button Should Not Be Selected	group_name	Verifies radio button group identified by group_name has no selection.
		See Select Radio Button for information about how radio buttons are located.
Register Keyword To Run On Failure	keyword	Sets the keyword to execute when a Selenium2Library keyword fails.
		keyword_name is the name of a keyword (from any available libraries) that will be executed if a Selenium2Library keyword fails. It is not possible to use a keyword that requires arguments. Using the value "Nothing" will disable this feature altogether.
		The initial keyword to use is set in importing, and the keyword that is used by default is Capture Page Screenshot. Taking a screenshot when something failed is a very useful feature, but notice that it can slow down the execution.
		This keyword returns the name of the previously registered failure keyword. It can be used to restore the original value later.
		Example:

Register Keyword To Run On Failure	Log Source	# Run Log Source on failure.	
${previous kw}=	Register Keyword To Run On Failure	Nothing	# Disables run-on-failure functionality and stores the previous kw name in a variable.
Register Keyword To Run On Failure	${previous kw}	# Restore to the previous keyword.	

Reload Page		Simulates user reloading page.
Remove Location Strategy	strategy_name	Removes a previously added custom location strategy.
		Will fail if a default strategy is specified.
		See Add Location Strategy for details about adding a custom location strategy.
Select All From List	locator	

		Selects all values from multi-select list identified by *id*.
		Key attributes for lists are *id* and *name*. See *introduction* for details about locating elements.
Select Checkbox	*locator*	Selects checkbox identified by *locator*.
		Does nothing if checkbox is already selected. Key attributes for checkboxes are *id* and *name*. See *introduction* for details about locating elements.
Select Frame	*locator*	Sets frame identified by *locator* as current frame.
		Key attributes for frames are *id* and *name*. See *introduction* for details about locating elements.
Select From List	*locator*, **items*	Selects **items* from list identified by *locator*
		If more than one value is given for a single-selection list, the last value will be selected. If the target list is a multi-selection list, and **items* is an empty list, all values of the list will be selected.
		*items try to select by value then by label.
		It's faster to use 'by index/value/label' functions.
		An exception is raised for a single-selection list if the last value does not exist in the list and a warning for all other non- existing items. For a multi-selection list, an exception is raised for any and all non-existing values.
		Select list keywords work on both lists and combo boxes. Key attributes for select lists are *id* and *name*. See *introduction* for details about locating elements.
Select From List By Index	*locator*, **indexes*	Selects **indexes* from list identified by *locator*
		Select list keywords work on both lists and combo boxes. Key attributes for select lists are *id* and *name*. See *introduction* for details about locating elements.
Select From List By Label	*locator*, **labels*	Selects **labels* from list identified by *locator*
		Select list keywords work on both lists and combo boxes. Key attributes for select lists are *id* and *name*. See *introduction* for details about locating elements.
Select From List By Value	*locator*, **values*	Selects **values* from list identified by *locator*
		Select list keywords work on both lists and combo boxes. Key attributes for select lists are *id* and *name*. See *introduction* for details about locating elements.
Select Radio Button	*group_name*, *value*	Sets selection of radio button group identified by *group_name* to *value*.

The radio button to be selected is located by two arguments:

- *group_name* is used as the name of the radio input
- *value* is used for the value attribute or for the id attribute

The XPath used to locate the correct radio button then looks like this: //input[@type='radio' and @name='group_name' and (@value='value' or @id='value')]

Examples:

Select Radio Button	size	XL	# Matches HTML like <input type="radio" name="size" value="XL">XL</input>
Select Radio Button	size	sizeXL	# Matches HTML like <input type="radio" name="size" value="XL" id="sizeXL">XL</input>

Select Window — *locator=None*

Selects the window matching locator and return previous window handle.

locator: any of name, title, url, window handle, excluded handle's list, or special words. return: either current window handle before selecting, or None if no current window.

If the window is found, all subsequent commands use that window, until this keyword is used again. If the window is not found, this keyword fails.

By default, when a locator value is provided, it is matched against the title of the window and the javascript name of the window. If multiple windows with same identifier are found, the first one is selected.

There are some special locators for searching target window: string 'main' (default): select the main window; string 'self': only return current window handle; string 'new': select the last-indexed window assuming it is the newest opened window window list: select the first window not in given list (See 'List Windows' to get the list)

It is also possible to specify the approach Selenium2Library should take to find a window by specifying a locator strategy:

Strategy	Example	Description
title	Select Window | title=My Document	Matches by window title
name	Select Window | name=${name}	Matches by window javascript name
url	Select Window | url=http://google.com	Matches by window's current URL

Example:

Click Link	popup_link	# opens new window	
Select Window	popupName		
Title Should Be	Popup Title		
Select Window			# Chooses the main window again

Set Browser Implicit Wait — *seconds*

Sets current browser's implicit wait in seconds.

From selenium 2 function 'Sets a sticky timeout to implicitly wait for an element to be found, or a command to complete. This method only needs to be called one time per session.'

Example:

Set Browser Implicit Wait	10 seconds

See also *Set Selenium Implicit Wait*.

Set Screenshot Directory — *path*, *persist=False*

Sets the root output directory for captured screenshots.

`path` argument specifies the absolute path where the screenshots should be written to. If the specified `path` does not exist, it will be created. Setting `persist` specifies that the given `path` should be used for the rest of the test execution, otherwise the path will be restored at the end of the currently executing scope.

Set Selenium Implicit Wait — *seconds*

Sets Selenium 2's default implicit wait in seconds and sets the implicit wait for all open browsers.

From selenium 2 function 'Sets a sticky timeout to implicitly wait for an element to be found, or a command to complete. This method only needs to be called one time per session.'

Example:

${orig wait} =	Set Selenium Implicit Wait	10 seconds
Perform AJAX call that is slow		
Set Selenium Implicit Wait	${orig wait}	

| **Set Selenium Speed** | *seconds* | Sets the delay in seconds that is waited after each Selenium command. |

This is useful mainly in slowing down the test execution to be able to view the execution. *seconds* may be given in Robot Framework time format. Returns the previous speed value in seconds.

One keyword may execute one or many Selenium commands and therefore one keyword may slow down more than the `seconds` argument defines. Example if delay is set to 1 second and because *Click Element* executes two Selenium commands, then the total delay will be 2 seconds. But because *Page Should Contain Element* executes only one selenium command, then the total delay will be 1 second.

Example:

| Set Selenium Speed | .5 seconds |

| **Set Selenium Timeout** | *seconds* | Sets the timeout in seconds used by various keywords. |

There are several *Wait* ... keywords that take timeout as an argument. All of these timeout arguments are optional. The timeout used by all of them can be set globally using this keyword. See *Timeouts* for more information about timeouts.

The previous timeout value is returned by this keyword and can be used to set the old value back later. The default timeout is 5 seconds, but it can be altered in *importing*.

Example:

${orig timeout} =		Set Selenium Timeout	15 seconds
Open page that loads slowly			
Set Selenium Timeout	${orig timeout}		

| **Set Window Position** | *x*, *y* | Sets the position x and y of the current window (relative to the left and top of the screen) to the specified values. |

Example:

Set Window Position	${8}	${10}
${x}	${y}=	Get Window Position
Should Be Equal	${x}	${8}
Should Be Equal	${y}	${10}

| **Set Window Size** | *width*, *height* | Sets the *width* and *height* of the current window to the specified values. |

Example:

Set Window Size	${800}	${600}
${width}	${height}=	Get Window Size
Should Be Equal	${width}	${800}
Should Be Equal	${height}	${600}

| **Simulate** | *locator*, *event* | Simulates *event* on element identified by *locator*. |

This keyword is useful if element has OnEvent handler that needs to be explicitly invoked.

See *introduction* for details about locating elements.

| **Submit Form** | *locator=None* | Submits a form identified by *locator*. |

If *locator* is empty, first form in the page will be submitted. Key attributes for forms are *id* and *name*. See *introduction* for details about locating elements.

| **Switch Browser** | *index_or_alias* | Switches between active browsers using index or alias. |

Index is returned from *Open Browser* and alias can be given to it.

Example:

Open Browser	http://google.com	ff	
Location Should Be	http://google.com		
Open Browser	http://yahoo.com	ie	2nd conn
Location Should Be	http://yahoo.com		
Switch Browser	1	# index	
Page Should Contain	I'm feeling lucky		
Switch Browser	2nd conn	# alias	
Page Should Contain	More Yahoo!		
Close All Browsers			

Above example expects that there was no other open browsers when opening the first one because it used index '1' when switching to it later. If you aren't sure about that you can store the index into a variable as below.

${id} =	Open Browser	http://google.com	*firefox
# Do something ...			
Switch Browser	${id}		

| **Table Cell Should Contain** | *table_locator*, *row*, *column*, *expected*, *loglevel=INFO* | Verifies that a certain cell in a table contains *expected*. |

Row and column number start from 1. This keyword passes if the specified cell contains the given content. If you want to test that the cell content matches exactly, or that it e.g. starts with some text, use *Get Table Cell* keyword in combination with built-in keywords such as *Should Be Equal* or *Should Start With*.

To understand how tables are identified, please take a look at the *introduction*.

See *Page Should Contain* for explanation about *loglevel* argument.

| **Table Column Should Contain** | *table_locator*, *col*, *expected*, *loglevel=INFO* | Verifies that a specific column contains *expected*. |

The first leftmost column is column number 1. A negative column number can be used to get column counting from the end of the row (end: -1). If the table contains cells that span multiple columns, those merged cells count as a single column. For example both tests below work, if in one row columns A and B are merged with colspan="2", and the logical third column contains "C".

Example:

| Table Column Should Contain | tableId | 3 | C |
| Table Column Should Contain | tableId | 2 | C |

To understand how tables are identified, please take a look at the *introduction*.

See *Page Should Contain Element* for explanation about *loglevel* argument.

Table Footer Should Contain	*table_locator*, *expected*, *loglevel=INFO*	Verifies that the table footer contains *expected*.
		With table footer can be described as any <td>-element that is child of a <tfoot>-element. To understand how tables are identified, please take a look at the *introduction*.
		See *Page Should Contain Element* for explanation about *loglevel* argument.
Table Header Should Contain	*table_locator*, *expected*, *loglevel=INFO*	Verifies that the table header, i.e. any <th>...</th> element, contains *expected*.
		To understand how tables are identified, please take a look at the *introduction*.
		See *Page Should Contain Element* for explanation about *loglevel* argument.
Table Row Should Contain	*table_locator*, *row*, *expected*, *loglevel=INFO*	Verifies that a specific table row contains *expected*.
		The uppermost row is row number 1. A negative column number can be used to get column counting from the end of the row (end: -1). For tables that are structured with thead, tbody and tfoot, only the tbody section is searched. Please use *Table Header Should Contain* or *Table Footer Should Contain* for tests against the header or footer content.
		If the table contains cells that span multiple rows, a match only occurs for the uppermost row of those merged cells. To understand how tables are identified, please take a look at the *introduction*.
		See *Page Should Contain Element* for explanation about *loglevel* argument.
Table Should Contain	*table_locator*, *expected*, *loglevel=INFO*	Verifies that *expected* can be found somewhere in the table.
		To understand how tables are identified, please take a look at the *introduction*.
		See *Page Should Contain Element* for explanation about *loglevel* argument.
Textarea Should Contain	*locator*, *expected*, *message=*	Verifies text area identified by *locator* contains text *expected*.
		message can be used to override default error message.
		Key attributes for text areas are *id* and *name*. See *introduction* for details about locating elements.
Textarea Value Should Be	*locator*, *expected*, *message=*	Verifies the value in text area identified by *locator* is exactly *expected*.
		message can be used to override default error message.
		Key attributes for text areas are *id* and *name*. See *introduction* for details about locating elements.
Textfield Should Contain	*locator*, *expected*, *message=*	Verifies text field identified by *locator* contains text *expected*.
		message can be used to override default error message.
		Key attributes for text fields are *id* and *name*. See *introduction* for details about locating elements.
Textfield Value Should Be	*locator*, *expected*, *message=*	Verifies the value in text field identified by *locator* is exactly *expected*.
		message can be used to override default error message.
		Key attributes for text fields are *id* and *name*. See *introduction* for details about locating elements.
Title Should Be	*title*	Verifies that current page title equals *title*.
Unselect Checkbox	*locator*	Removes selection of checkbox identified by *locator*.
		Does nothing if the checkbox is not checked. Key attributes for checkboxes are *id* and *name*. See *introduction* for details about locating elements.
Unselect Frame		Sets the top frame as the current frame.
Unselect From List	*locator*, *items*	Unselects given values from select list identified by locator.
		As a special case, giving empty list as *items* will remove all selections.
		*items try to unselect by value AND by label.
		It's faster to use 'by index/value/label' functions.
		Select list keywords work on both lists and combo boxes. Key attributes for select lists are *id* and *name*. See *introduction* for details about locating elements.
Unselect From List By Index	*locator*, *indexes*	Unselects *indexes* from list identified by *locator*
		Select list keywords work on both lists and combo boxes. Key attributes for select lists are *id* and *name*. See *introduction* for details about locating elements.
Unselect From List By Label	*locator*, *labels*	Unselects *labels* from list identified by *locator*
		Select list keywords work on both lists and combo boxes. Key attributes for select lists are *id* and *name*. See *introduction* for details about locating elements.
Unselect From List By Value	*locator*, *values*	Unselects *values* from list identified by *locator*
		Select list keywords work on both lists and combo boxes. Key attributes for select lists are *id* and *name*. See *introduction* for details about locating elements.
Wait For Condition	*condition*, *timeout=None*, *error=None*	Waits until the given *condition* is true or *timeout* expires.
		The *condition* can be arbitrary JavaScript expression but must contain a return statement (with the value to be returned) at the end. See *Execute JavaScript* for information about accessing the actual contents of the window through JavaScript.
		error can be used to override the default error message.
		See *introduction* for more information about *timeout* and its default value.
		See also *Wait Until Page Contains*, *Wait Until Page Contains Element*, *Wait Until Element Is Visible* and BuiltIn keyword *Wait Until Keyword Succeeds*.
Wait Until Element Contains	*locator*, *text*, *timeout=None*, *error=None*	Waits until given element contains *text*.
		Fails if *timeout* expires before the text appears on given element. See *introduction* for more information about *timeout* and its default value.
		error can be used to override the default error message.
		See also *Wait Until Page Contains*, *Wait Until Page Contains Element*, *Wait For Condition*, *Wait Until Element Is Visible* and BuiltIn keyword *Wait Until Keyword Succeeds*.
Wait Until Element Does Not Contain	*locator*, *text*, *timeout=None*, *error=None*	Waits until given element does not contain *text*.
		Fails if *timeout* expires before the text disappears from given element. See *introduction* for more information about *timeout* and its default value.

		error can be used to override the default error message.
		See also *Wait Until Page Contains*, *Wait Until Page Contains Element*, *Wait For Condition*, *Wait Until Element Is Visible* and BuiltIn keyword *Wait Until Keyword Succeeds*.
Wait Until Element Is Enabled	*locator*, *timeout=None*, *error=None*	Waits until element specified with *locator* is enabled.
		Fails if *timeout* expires before the element is enabled. See *introduction* for more information about *timeout* and its default value.
		error can be used to override the default error message.
		See also *Wait Until Page Contains*, *Wait Until Page Contains Element*, *Wait For Condition* and BuiltIn keyword *Wait Until Keyword Succeeds*.
Wait Until Element Is Not Visible	*locator*, *timeout=None*, *error=None*	Waits until element specified with *locator* is not visible.
		Fails if *timeout* expires before the element is not visible. See *introduction* for more information about *timeout* and its default value.
		error can be used to override the default error message.
		See also *Wait Until Page Contains*, *Wait Until Page Contains Element*, *Wait For Condition* and BuiltIn keyword *Wait Until Keyword Succeeds*.
Wait Until Element Is Visible	*locator*, *timeout=None*, *error=None*	Waits until element specified with *locator* is visible.
		Fails if *timeout* expires before the element is visible. See *introduction* for more information about *timeout* and its default value.
		error can be used to override the default error message.
		See also *Wait Until Page Contains*, *Wait Until Page Contains Element*, *Wait For Condition* and BuiltIn keyword *Wait Until Keyword Succeeds*.
Wait Until Page Contains	*text*, *timeout=None*, *error=None*	Waits until *text* appears on current page.
		Fails if *timeout* expires before the text appears. See *introduction* for more information about *timeout* and its default value.
		error can be used to override the default error message.
		See also *Wait Until Page Contains Element*, *Wait For Condition*, *Wait Until Element Is Visible* and BuiltIn keyword *Wait Until Keyword Succeeds*.
Wait Until Page Contains Element	*locator*, *timeout=None*, *error=None*	Waits until element specified with *locator* appears on current page.
		Fails if *timeout* expires before the element appears. See *introduction* for more information about *timeout* and its default value.
		error can be used to override the default error message.
		See also *Wait Until Page Contains*, *Wait For Condition*, *Wait Until Element Is Visible* and BuiltIn keyword *Wait Until Keyword Succeeds*.
Wait Until Page Does Not Contain	*text*, *timeout=None*, *error=None*	Waits until *text* disappears from current page.
		Fails if *timeout* expires before the *text* disappears. See *introduction* for more information about *timeout* and its default value.
		error can be used to override the default error message.
		See also *Wait Until Page Contains*, *Wait For Condition*, *Wait Until Element Is Visible* and BuiltIn keyword *Wait Until Keyword Succeeds*.
Wait Until Page Does Not Contain Element	*locator*, *timeout=None*, *error=None*	Waits until element specified with *locator* disappears from current page.
		Fails if *timeout* expires before the element disappears. See *introduction* for more information about *timeout* and its default value.
		error can be used to override the default error message.
		See also *Wait Until Page Contains*, *Wait For Condition*, *Wait Until Element Is Visible* and BuiltIn keyword *Wait Until Keyword Succeeds*.
Xpath Should Match X Times	*xpath*, *expected_xpath_count*, *message=*, *loglevel=INFO*	Verifies that the page contains the given number of elements located by the given *xpath*.
		One should not use the xpath= prefix for 'xpath'. XPath is assumed.
		Correct:
		` Xpath Should Match X Times	//div[@id='sales-pop']	1`
		Incorrect:
		` Xpath Should Match X Times	xpath=//div[@id='sales-pop']	1`
		See *Page Should Contain Element* for explanation about *message* and *loglevel* arguments.

Altogether 167 keywords.
Generated by Libdoc on 2017-09-01 21:47:03.