**WebDriver**

New generation Selenium WebDriver module.

**Local Testing**

**Selenium**

To run Selenium Server you need [Java](https://www.java.com/) as well as Chrome or Firefox browser installed.

1. Download [Selenium Standalone Server](http://docs.seleniumhq.org/download/)
2. To use Chrome, install [ChromeDriver](https://sites.google.com/a/chromium.org/chromedriver/getting-started). To use Firefox, install [GeckoDriver](https://github.com/mozilla/geckodriver).
3. Launch the Selenium Server: java -jar selenium-server-standalone-3.xx.xxx.jar. To locate Chromedriver binary use -Dwebdriver.chrome.driver=./chromedriver option. For Geckodriver use -Dwebdriver.gecko.driver=./geckodriver.
4. Configure this module (in acceptance.suite.yml) by setting url and browser:

modules:  
 enabled:  
 - WebDriver:  
 url: 'http://localhost/'  
 browser: chrome # 'chrome' or 'firefox'

Launch Selenium Server before executing tests.

java -jar "/path/to/selenium-server-standalone-xxx.jar"

**ChromeDriver**

To run tests in Chrome browser you may connect to ChromeDriver directly, without using Selenium Server.

1. Install [ChromeDriver](https://sites.google.com/a/chromium.org/chromedriver/getting-started).
2. Launch ChromeDriver: chromedriver --url-base=/wd/hub
3. Configure this module to use ChromeDriver port:

modules:  
 enabled:  
 - WebDriver:  
 url: 'http://localhost/'  
 window\_size: false # disabled in ChromeDriver  
 port: 9515  
 browser: chrome  
 capabilities:  
 chromeOptions: # additional chrome options

Additional [Chrome options](https://sites.google.com/a/chromium.org/chromedriver/capabilities) can be set in chromeOptions capabilities.

**PhantomJS**

PhantomJS is a [headless browser](https://en.wikipedia.org/wiki/Headless_browser) alternative to Selenium Server that implements [the WebDriver protocol](https://code.google.com/p/selenium/wiki/JsonWireProtocol). It allows you to run Selenium tests on a server without a GUI installed.

1. Download [PhantomJS](http://phantomjs.org/download.html)
2. Run PhantomJS in WebDriver mode: phantomjs --webdriver=4444
3. Configure this module (in acceptance.suite.yml) by setting url and phantomjs as browser:

modules:  
 enabled:  
 - WebDriver:  
 url: 'http://localhost/'  
 browser: phantomjs

Since PhantomJS doesn’t give you any visual feedback, it’s probably a good idea to install [Codeception\Extension\Recorder](http://codeception.com/extensions#CodeceptionExtensionRecorder) which gives you screenshots of how PhantomJS “sees” your pages.

**Headless Selenium in Docker**

Docker can ship Selenium Server with all its dependencies and browsers inside a single container. Running tests inside Docker is as easy as pulling [official selenium image](https://github.com/SeleniumHQ/docker-selenium) and starting a container with Chrome:

docker run --net=host selenium/standalone-chrome

By using --net=host we allow selenium to access local websites.

**Cloud Testing**

Cloud Testing services can run your WebDriver tests in the cloud. In case you want to test a local site or site behind a firewall you should use a tunnel application provided by a service.

**SauceLabs**

1. Create an account at [SauceLabs.com](http://saucelabs.com/) to get your username and access key
2. In the module configuration use the format username:access\_key@ondemand.saucelabs.com’ for host
3. Configure platform under capabilities to define the [Operating System](https://docs.saucelabs.com/reference/platforms-configurator/#/)
4. run a tunnel app if your site can’t be accessed from Internet

modules:  
 enabled:  
 - WebDriver:  
 url: http://mysite.com  
 host: '<username>:<access key>@ondemand.saucelabs.com'  
 port: 80  
 browser: chrome  
 capabilities:  
 platform: 'Windows 10'

**BrowserStack**

1. Create an account at [BrowserStack](https://www.browserstack.com/) to get your username and access key
2. In the module configuration use the format username:access\_key@hub.browserstack.com’ for host
3. Configure os and os\_version under capabilities to define the operating System
4. If your site is available only locally or via VPN you should use a tunnel app. In this case add browserstack.local capability and set it to true.

modules:  
 enabled:  
 - WebDriver:  
 url: http://mysite.com  
 host: '<username>:<access key>@hub.browserstack.com'  
 port: 80  
 browser: chrome  
 capabilities:  
 os: Windows  
 os\_version: 10  
 browserstack.local: true # for local testing

**TestingBot**

1. Create an account at [TestingBot](https://testingbot.com/) to get your key and secret
2. In the module configuration use the format key:secret@hub.testingbot.com’ for host
3. Configure platform under capabilities to define the [Operating System](https://testingbot.com/support/getting-started/browsers.html)
4. Run [TestingBot Tunnel](https://testingbot.com/support/other/tunnel) if your site can’t be accessed from Internet

modules:  
 enabled:  
 - WebDriver:  
 url: http://mysite.com  
 host: '<key>:<secret>@hub.testingbot.com'  
 port: 80  
 browser: chrome  
 capabilities:  
 platform: Windows 10

**Configuration**

* url required - Starting URL for your app.
* browser required - Browser to launch.
* host - Selenium server host (127.0.0.1 by default).
* port - Selenium server port (4444 by default).
* restart - Set to false (default) to use the same browser window for all tests, or set to true to create a new window for each test. In any case, when all tests are finished the browser window is closed.
* start - Autostart a browser for tests. Can be disabled if browser session is started with \_initializeSession inside a Helper.
* window\_size - Initial window size. Set to maximize or a dimension in the format 640x480.
* clear\_cookies - Set to false to keep cookies, or set to true (default) to delete all cookies between tests.
* wait (default: 0 seconds) - Whenever element is required and is not on page, wait for n seconds to find it before fail.
* capabilities - Sets Selenium [desired capabilities](https://github.com/SeleniumHQ/selenium/wiki/DesiredCapabilities). Should be a key-value array.
* connection\_timeout - timeout for opening a connection to remote selenium server (30 seconds by default).
* request\_timeout - timeout for a request to return something from remote selenium server (30 seconds by default).
* pageload\_timeout - amount of time to wait for a page load to complete before throwing an error (default 0 seconds).
* http\_proxy - sets http proxy server url for testing a remote server.
* http\_proxy\_port - sets http proxy server port
* debug\_log\_entries - how many selenium entries to print with debugWebDriverLogs or on fail (15 by default).
* log\_js\_errors - Set to true to include possible JavaScript to HTML report, or set to false (default) to deactivate.

Example (acceptance.suite.yml)

modules:  
 enabled:  
 - WebDriver:  
 url: 'http://localhost/'  
 browser: firefox  
 window\_size: 1024x768  
 capabilities:  
 unexpectedAlertBehaviour: 'accept'  
 firefox\_profile: '~/firefox-profiles/codeception-profile.zip.b64'

**Usage**

**Locating Elements**

Most methods in this module that operate on a DOM element (e.g. click) accept a locator as the first argument, which can be either a string or an array.

If the locator is an array, it should have a single element, with the key signifying the locator type (id, name, css, xpath, link, or class) and the value being the locator itself. This is called a “strict” locator. Examples:

* ['id' => 'foo'] matches <div id="foo">
* ['name' => 'foo'] matches <div name="foo">
* ['css' => 'input[type=input][value=foo]'] matches <input type="input" value="foo">
* ['xpath' => "//input[@type='submit'][contains(@value, 'foo')]"] matches <input type="submit" value="foobar">
* ['link' => 'Click here'] matches <a href="google.com">Click here</a>
* ['class' => 'foo'] matches <div class="foo">

Writing good locators can be tricky. The Mozilla team has written an excellent guide titled [Writing reliable locators for Selenium and WebDriver tests](https://blog.mozilla.org/webqa/2013/09/26/writing-reliable-locators-for-selenium-and-webdriver-tests/).

If you prefer, you may also pass a string for the locator. This is called a “fuzzy” locator. In this case, Codeception uses a a variety of heuristics (depending on the exact method called) to determine what element you’re referring to. For example, here’s the heuristic used for the submitForm method:

1. Does the locator look like an ID selector (e.g. “#foo”)? If so, try to find a form matching that ID.
2. If nothing found, check if locator looks like a CSS selector. If so, run it.
3. If nothing found, check if locator looks like an XPath expression. If so, run it.
4. Throw an ElementNotFound exception.

Be warned that fuzzy locators can be significantly slower than strict locators. Especially if you use Selenium WebDriver with wait (aka implicit wait) option. In the example above if you set wait to 5 seconds and use XPath string as fuzzy locator, submitForm method will wait for 5 seconds at each step. That means 5 seconds finding the form by ID, another 5 seconds finding by CSS until it finally tries to find the form by XPath). If speed is a concern, it’s recommended you stick with explicitly specifying the locator type via the array syntax.

**Public Properties**

* webDriver - instance of \Facebook\WebDriver\Remote\RemoteWebDriver. Can be accessed from Helper classes for complex WebDriver interactions.

// inside Helper class  
$this->getModule('WebDriver')->webDriver->getKeyboard()->sendKeys('hello, webdriver');

**Actions**

**\_backupSession**

hidden API method, expected to be used from Helper classes

Returns current WebDriver session for saving

* return RemoteWebDriver

**\_capabilities**

hidden API method, expected to be used from Helper classes

Change capabilities of WebDriver. Should be executed before starting a new browser session. This method expects a function to be passed which returns array or [WebDriver Desired Capabilities](https://github.com/facebook/php-webdriver/blob/community/lib/Remote/DesiredCapabilities.php) object. Additional [Chrome options](https://github.com/facebook/php-webdriver/wiki/ChromeOptions) (like adding extensions) can be passed as well.

<?php // in helper  
public function \_before(TestInterface $test)  
{  
 $this->getModule('WebDriver')->\_capabilities(function($currentCapabilities) {  
 // or new \Facebook\WebDriver\Remote\DesiredCapabilities();  
 return \Facebook\WebDriver\Remote\DesiredCapabilities::firefox();  
 });  
}

to make this work load \Helper\Acceptance before WebDriver in acceptance.suite.yml:

modules:  
 enabled:  
 - \Helper\Acceptance  
 - WebDriver

For instance, [**BrowserStack** cloud service](https://www.browserstack.com/automate/capabilities) may require a test name to be set in capabilities. This is how it can be done via \_capabilities method from Helper\Acceptance:

<?php // inside Helper\Acceptance  
public function \_before(TestInterface $test)  
{  
 $name = $test->getMetadata()->getName();  
 $this->getModule('WebDriver')->\_capabilities(function($currentCapabilities) use ($name) {  
 $currentCapabilities['name'] = $name;  
 return $currentCapabilities;  
 });  
}

In this case, please ensure that \Helper\Acceptance is loaded before WebDriver so new capabilities could be applied.

* param \Closure $capabilityFunction

**\_closeSession**

hidden API method, expected to be used from Helper classes

Manually closes current WebDriver session.

<?php  
$this->getModule('WebDriver')->\_closeSession();  
  
// close a specific session  
$webDriver = $this->getModule('WebDriver')->webDriver;  
$this->getModule('WebDriver')->\_closeSession($webDriver);

* param $webDriver (optional) a specific webdriver session instance

**\_findClickable**

hidden API method, expected to be used from Helper classes

Locates a clickable element.

Use it in Helpers or GroupObject or Extension classes:

<?php  
$module = $this->getModule('WebDriver');  
$page = $module->webDriver;  
  
// search a link or button on a page  
$el = $module->\_findClickable($page, 'Click Me');  
  
// search a link or button within an element  
$topBar = $module->\_findElements('.top-bar')[0];  
$el = $module->\_findClickable($topBar, 'Click Me');

* param $page WebDriver instance or an element to search within
* param $link a link text or locator to click
* return WebDriverElement

**\_findElements**

hidden API method, expected to be used from Helper classes

Locates element using available Codeception locator types:

* XPath
* CSS
* Strict Locator

Use it in Helpers or GroupObject or Extension classes:

<?php  
$els = $this->getModule('WebDriver')->\_findElements('.items');  
$els = $this->getModule('WebDriver')->\_findElements(['name' => 'username']);  
  
$editLinks = $this->getModule('WebDriver')->\_findElements(['link' => 'Edit']);  
// now you can iterate over $editLinks and check that all them have valid hrefs

WebDriver module returns Facebook\WebDriver\Remote\RemoteWebElement instances PhpBrowser and Framework modules return Symfony\Component\DomCrawler\Crawler instances

* param $locator
* return array of interactive elements

**\_getCurrentUri**

hidden API method, expected to be used from Helper classes

Uri of currently opened page.

* return string @throws ModuleException

**\_getUrl**

hidden API method, expected to be used from Helper classes

Returns URL of a host.

@throws ModuleConfigException

**\_initializeSession**

hidden API method, expected to be used from Helper classes

Manually starts a new browser session.

<?php  
$this->getModule('WebDriver')->\_initializeSession();

**\_loadSession**

hidden API method, expected to be used from Helper classes

Loads current RemoteWebDriver instance as a session

* param RemoteWebDriver $session

**\_restart**

hidden API method, expected to be used from Helper classes

Restarts a web browser. Can be used with \_reconfigure to open browser with different configuration

<?php  
// inside a Helper  
$this->getModule('WebDriver')->\_restart(); // just restart  
$this->getModule('WebDriver')->\_restart(['browser' => $browser]); // reconfigure + restart

* param array $config

**\_savePageSource**

hidden API method, expected to be used from Helper classes

Saves HTML source of a page to a file

* param $filename

**\_saveScreenshot**

hidden API method, expected to be used from Helper classes

Saves screenshot of current page to a file

$this->getModule('WebDriver')->\_saveScreenshot(codecept\_output\_dir().'screenshot\_1.png');

* param $filename

**acceptPopup**

Accepts the active JavaScript native popup window, as created by window.alert|window.confirm|window.prompt. Don’t confuse popups with modal windows, as created by [various libraries](http://jster.net/category/windows-modals-popups).

**amOnPage**

Opens the page for the given relative URI.

<?php  
// opens front page  
$I->amOnPage('/');  
// opens /register page  
$I->amOnPage('/register');

* param string $page

**amOnSubdomain**

Changes the subdomain for the ‘url’ configuration parameter. Does not open a page; use amOnPage for that.

<?php  
// If config is: 'http://mysite.com'  
// or config is: 'http://www.mysite.com'  
// or config is: 'http://company.mysite.com'  
  
$I->amOnSubdomain('user');  
$I->amOnPage('/');  
// moves to http://user.mysite.com/  
?>

* param $subdomain

**amOnUrl**

Open web page at the given absolute URL and sets its hostname as the base host.

<?php  
$I->amOnUrl('http://codeception.com');  
$I->amOnPage('/quickstart'); // moves to http://codeception.com/quickstart  
?>

**appendField**

Append the given text to the given element. Can also add a selection to a select box.

<?php  
$I->appendField('#mySelectbox', 'SelectValue');  
$I->appendField('#myTextField', 'appended');  
?>

* param string $field
* param string $value @throws \Codeception\Exception\ElementNotFound

**attachFile**

Attaches a file relative to the Codeception \_data directory to the given file upload field.

<?php  
// file is stored in 'tests/\_data/prices.xls'  
$I->attachFile('input[@type="file"]', 'prices.xls');  
?>

* param $field
* param $filename

**cancelPopup**

Dismisses the active JavaScript popup, as created by window.alert, window.confirm, or window.prompt.

**checkOption**

Ticks a checkbox. For radio buttons, use the selectOption method instead.

<?php  
$I->checkOption('#agree');  
?>

* param $option

**click**

Perform a click on a link or a button, given by a locator. If a fuzzy locator is given, the page will be searched for a button, link, or image matching the locator string. For buttons, the “value” attribute, “name” attribute, and inner text are searched. For links, the link text is searched. For images, the “alt” attribute and inner text of any parent links are searched.

The second parameter is a context (CSS or XPath locator) to narrow the search.

Note that if the locator matches a button of type submit, the form will be submitted.

<?php  
// simple link  
$I->click('Logout');  
// button of form  
$I->click('Submit');  
// CSS button  
$I->click('#form input[type=submit]');  
// XPath  
$I->click('//form/\*[@type=submit]');  
// link in context  
$I->click('Logout', '#nav');  
// using strict locator  
$I->click(['link' => 'Login']);  
?>

* param $link
* param $context

**clickWithLeftButton**

Performs click with the left mouse button on an element. If the first parameter null then the offset is relative to the actual mouse position. If the second and third parameters are given, then the mouse is moved to an offset of the element’s top-left corner. Otherwise, the mouse is moved to the center of the element.

<?php  
$I->clickWithLeftButton(['css' => '.checkout']);  
$I->clickWithLeftButton(null, 20, 50);  
$I->clickWithLeftButton(['css' => '.checkout'], 20, 50);  
?>

* param string $cssOrXPath css or xpath of the web element (body by default).
* param int $offsetX
* param int $offsetY

@throws \Codeception\Exception\ElementNotFound

**clickWithRightButton**

Performs contextual click with the right mouse button on an element. If the first parameter null then the offset is relative to the actual mouse position. If the second and third parameters are given, then the mouse is moved to an offset of the element’s top-left corner. Otherwise, the mouse is moved to the center of the element.

<?php  
$I->clickWithRightButton(['css' => '.checkout']);  
$I->clickWithRightButton(null, 20, 50);  
$I->clickWithRightButton(['css' => '.checkout'], 20, 50);  
?>

* param string $cssOrXPath css or xpath of the web element (body by default).
* param int $offsetX
* param int $offsetY

@throws \Codeception\Exception\ElementNotFound

**closeTab**

Closes current browser tab and switches to previous active tab.

<?php  
$I->closeTab();

Can’t be used with PhantomJS

**debugWebDriverLogs**

Print out latest Selenium Logs in debug mode

* param TestInterface $test

**dontSee**

Checks that the current page doesn’t contain the text specified (case insensitive). Give a locator as the second parameter to match a specific region.

<?php  
$I->dontSee('Login'); // I can suppose user is already logged in  
$I->dontSee('Sign Up','h1'); // I can suppose it's not a signup page  
$I->dontSee('Sign Up','//body/h1'); // with XPath  
$I->dontSee('Sign Up', ['css' => 'body h1']); // with strict CSS locator

Note that the search is done after stripping all HTML tags from the body, so $I->dontSee('strong') will fail on strings like:

* <p>I am Stronger than thou</p>
* <script>document.createElement('strong');</script>

But will ignore strings like:

* <strong>Home</strong>
* <div class="strong">Home</strong>
* <!-- strong -->

For checking the raw source code, use seeInSource().

* param string $text
* param string $selector optional

**dontSeeCheckboxIsChecked**

Check that the specified checkbox is unchecked.

<?php  
$I->dontSeeCheckboxIsChecked('#agree'); // I suppose user didn't agree to terms  
$I->seeCheckboxIsChecked('#signup\_form input[type=checkbox]'); // I suppose user didn't check the first checkbox in form.  
?>

* param $checkbox

**dontSeeCookie**

Checks that there isn’t a cookie with the given name. You can set additional cookie params like domain, path as array passed in last argument.

* param $cookie
* param array $params

**dontSeeCurrentUrlEquals**

Checks that the current URL doesn’t equal the given string. Unlike dontSeeInCurrentUrl, this only matches the full URL.

<?php  
// current url is not root  
$I->dontSeeCurrentUrlEquals('/');  
?>

* param string $uri

**dontSeeCurrentUrlMatches**

Checks that current url doesn’t match the given regular expression.

<?php  
// to match root url  
$I->dontSeeCurrentUrlMatches('~$/users/(\d+)~');  
?>

* param string $uri

**dontSeeElement**

Checks that the given element is invisible or not present on the page. You can also specify expected attributes of this element.

<?php  
$I->dontSeeElement('.error');  
$I->dontSeeElement('//form/input[1]');  
$I->dontSeeElement('input', ['name' => 'login']);  
$I->dontSeeElement('input', ['value' => '123456']);  
?>

* param $selector
* param array $attributes

**dontSeeElementInDOM**

Opposite of seeElementInDOM.

* param $selector
* param array $attributes

**dontSeeInCurrentUrl**

Checks that the current URI doesn’t contain the given string.

<?php  
$I->dontSeeInCurrentUrl('/users/');  
?>

* param string $uri

**dontSeeInField**

Checks that an input field or textarea doesn’t contain the given value. For fuzzy locators, the field is matched by label text, CSS and XPath.

<?php  
$I->dontSeeInField('Body','Type your comment here');  
$I->dontSeeInField('form textarea[name=body]','Type your comment here');  
$I->dontSeeInField('form input[type=hidden]','hidden\_value');  
$I->dontSeeInField('#searchform input','Search');  
$I->dontSeeInField('//form/\*[@name=search]','Search');  
$I->dontSeeInField(['name' => 'search'], 'Search');  
?>

* param $field
* param $value

**dontSeeInFormFields**

Checks if the array of form parameters (name => value) are not set on the form matched with the passed selector.

<?php  
$I->dontSeeInFormFields('form[name=myform]', [  
 'input1' => 'non-existent value',  
 'input2' => 'other non-existent value',  
]);  
?>

To check that an element hasn’t been assigned any one of many values, an array can be passed as the value:

<?php  
$I->dontSeeInFormFields('.form-class', [  
 'fieldName' => [  
 'This value shouldn\'t be set',  
 'And this value shouldn\'t be set',  
 ],  
]);  
?>

Additionally, checkbox values can be checked with a boolean.

<?php  
$I->dontSeeInFormFields('#form-id', [  
 'checkbox1' => true, // fails if checked  
 'checkbox2' => false, // fails if unchecked  
]);  
?>

* param $formSelector
* param $params

**dontSeeInPageSource**

Checks that the page source doesn’t contain the given string.

* param $text

**dontSeeInPopup**

Checks that the active JavaScript popup, as created by window.alert|window.confirm|window.prompt, does NOT contain the given string.

* param $text

@throws \Codeception\Exception\ModuleException

**dontSeeInSource**

Checks that the current page contains the given string in its raw source code.

<?php  
$I->dontSeeInSource('<h1>Green eggs &amp; ham</h1>');

* param $raw

**dontSeeInTitle**

Checks that the page title does not contain the given string.

* param $title

**dontSeeLink**

Checks that the page doesn’t contain a link with the given string. If the second parameter is given, only links with a matching “href” attribute will be checked.

<?php  
$I->dontSeeLink('Logout'); // I suppose user is not logged in  
$I->dontSeeLink('Checkout now', '/store/cart.php');  
?>

* param string $text
* param string $url optional

**dontSeeOptionIsSelected**

Checks that the given option is not selected.

<?php  
$I->dontSeeOptionIsSelected('#form input[name=payment]', 'Visa');  
?>

* param $selector
* param $optionText

**doubleClick**

Performs a double-click on an element matched by CSS or XPath.

* param $cssOrXPath @throws \Codeception\Exception\ElementNotFound

**dragAndDrop**

Performs a simple mouse drag-and-drop operation.

<?php  
$I->dragAndDrop('#drag', '#drop');  
?>

* param string $source (CSS ID or XPath)
* param string $target (CSS ID or XPath)

**executeInSelenium**

Low-level API method. If Codeception commands are not enough, this allows you to use Selenium WebDriver methods directly:

$I->executeInSelenium(function(\Facebook\WebDriver\Remote\RemoteWebDriver $webdriver) {  
 $webdriver->get('http://google.com');  
});

This runs in the context of the [RemoteWebDriver class](https://github.com/facebook/php-webdriver/blob/master/lib/remote/RemoteWebDriver.php). Try not to use this command on a regular basis. If Codeception lacks a feature you need, please implement it and submit a patch.

* param callable $function

**executeJS**

Executes custom JavaScript.

This example uses jQuery to get a value and assigns that value to a PHP variable:

<?php  
$myVar = $I->executeJS('return $("#myField").val()');  
?>

* param $script

**fillField**

Fills a text field or textarea with the given string.

<?php  
$I->fillField("//input[@type='text']", "Hello World!");  
$I->fillField(['name' => 'email'], 'jon@mail.com');  
?>

* param $field
* param $value

**grabAttributeFrom**

Grabs the value of the given attribute value from the given element. Fails if element is not found.

<?php  
$I->grabAttributeFrom('#tooltip', 'title');  
?>

* param $cssOrXpath
* param $attribute

**grabCookie**

Grabs a cookie value. You can set additional cookie params like domain, path in array passed as last argument.

* param $cookie
* param array $params

**grabFromCurrentUrl**

Executes the given regular expression against the current URI and returns the first capturing group. If no parameters are provided, the full URI is returned.

<?php  
$user\_id = $I->grabFromCurrentUrl('~$/user/(\d+)/~');  
$uri = $I->grabFromCurrentUrl();  
?>

* param string $uri optional

**grabMultiple**

Grabs either the text content, or attribute values, of nodes matched by $cssOrXpath and returns them as an array.

<a href="#first">First</a>  
<a href="#second">Second</a>  
<a href="#third">Third</a>

<?php  
// would return ['First', 'Second', 'Third']  
$aLinkText = $I->grabMultiple('a');  
  
// would return ['#first', '#second', '#third']  
$aLinks = $I->grabMultiple('a', 'href');  
?>

* param $cssOrXpath
* param $attribute
* return string[]

**grabPageSource**

Grabs current page source code.

@throws ModuleException if no page was opened.

* return string Current page source code.

**grabTextFrom**

Finds and returns the text contents of the given element. If a fuzzy locator is used, the element is found using CSS, XPath, and by matching the full page source by regular expression.

<?php  
$heading = $I->grabTextFrom('h1');  
$heading = $I->grabTextFrom('descendant-or-self::h1');  
$value = $I->grabTextFrom('~<input value=(.\*?)]~sgi'); // match with a regex  
?>

* param $cssOrXPathOrRegex

**grabValueFrom**

Finds the value for the given form field. If a fuzzy locator is used, the field is found by field name, CSS, and XPath.

<?php  
$name = $I->grabValueFrom('Name');  
$name = $I->grabValueFrom('input[name=username]');  
$name = $I->grabValueFrom('descendant-or-self::form/descendant::input[@name = 'username']');  
$name = $I->grabValueFrom(['name' => 'username']);  
?>

* param $field

**loadSessionSnapshot**

* param string $name
* return bool

**makeScreenshot**

Takes a screenshot of the current window and saves it to tests/\_output/debug.

<?php  
$I->amOnPage('/user/edit');  
$I->makeScreenshot('edit\_page');  
// saved to: tests/\_output/debug/edit\_page.png  
$I->makeScreenshot();  
// saved to: tests/\_output/debug/2017-05-26\_14-24-11\_4b3403665fea6.png

* param $name

**maximizeWindow**

Maximizes the current window.

**moveBack**

Moves back in history.

**moveForward**

Moves forward in history.

**moveMouseOver**

Move mouse over the first element matched by the given locator. If the first parameter null then the page is used. If the second and third parameters are given, then the mouse is moved to an offset of the element’s top-left corner. Otherwise, the mouse is moved to the center of the element.

<?php  
$I->moveMouseOver(['css' => '.checkout']);  
$I->moveMouseOver(null, 20, 50);  
$I->moveMouseOver(['css' => '.checkout'], 20, 50);  
?>

* param string $cssOrXPath css or xpath of the web element
* param int $offsetX
* param int $offsetY

@throws \Codeception\Exception\ElementNotFound

**openNewTab**

Opens a new browser tab (wherever it is possible) and switches to it.

<?php  
$I->openNewTab();

Tab is opened by using window.open javascript in a browser. Please note, that adblock can restrict creating such tabs.

Can’t be used with PhantomJS

**pauseExecution**

Pauses test execution in debug mode. To proceed test press “ENTER” in console.

This method is useful while writing tests, since it allows you to inspect the current page in the middle of a test case.

**performOn**

Waits for element and runs a sequence of actions inside its context. Actions can be defined with array, callback, or Codeception\Util\ActionSequence instance.

Actions as array are recommended for simple to combine “waitForElement” with assertions; waitForElement($el) and see('text', $el) can be simplified to:

<?php  
$I->performOn($el, ['see' => 'text']);

List of actions can be pragmatically build using Codeception\Util\ActionSequence:

<?php  
$I->performOn('.model', ActionSequence::build()  
 ->see('Warning')  
 ->see('Are you sure you want to delete this?')  
 ->click('Yes')  
);

Actions executed from array or ActionSequence will print debug output for actions, and adds an action name to exception on failure.

Whenever you need to define more actions a callback can be used. A WebDriver module is passed for argument:

<?php  
$I->performOn('.rememberMe', function (WebDriver $I) {  
 $I->see('Remember me next time');  
 $I->seeElement('#LoginForm\_rememberMe');  
 $I->dontSee('Login');  
});

In 3rd argument you can set number a seconds to wait for element to appear

* param $element
* param $actions
* param int $timeout

**pressKey**

Presses the given key on the given element. To specify a character and modifier (e.g. ctrl, alt, shift, meta), pass an array for $char with the modifier as the first element and the character as the second. For special keys use key constants from WebDriverKeys class.

<?php  
// <input id="page" value="old" />  
$I->pressKey('#page','a'); // => olda  
$I->pressKey('#page',array('ctrl','a'),'new'); //=> new  
$I->pressKey('#page',array('shift','111'),'1','x'); //=> old!!!1x  
$I->pressKey('descendant-or-self::\*[@id='page']','u'); //=> oldu  
$I->pressKey('#name', array('ctrl', 'a'), \Facebook\WebDriver\WebDriverKeys::DELETE); //=>''  
?>

* param $element
* param $char string|array Can be char or array with modifier. You can provide several chars. @throws \Codeception\Exception\ElementNotFound

**reloadPage**

Reloads the current page.

**resetCookie**

Unsets cookie with the given name. You can set additional cookie params like domain, path in array passed as last argument.

* param $cookie
* param array $params

**resizeWindow**

Resize the current window.

<?php  
$I->resizeWindow(800, 600);

* param int $width
* param int $height

**saveSessionSnapshot**

* param string $name

**scrollTo**

Move to the middle of the given element matched by the given locator. Extra shift, calculated from the top-left corner of the element, can be set by passing $offsetX and $offsetY parameters.

<?php  
$I->scrollTo(['css' => '.checkout'], 20, 50);  
?>

* param $selector
* param int $offsetX
* param int $offsetY

**see**

Checks that the current page contains the given string (case insensitive).

You can specify a specific HTML element (via CSS or XPath) as the second parameter to only search within that element.

<?php  
$I->see('Logout'); // I can suppose user is logged in  
$I->see('Sign Up', 'h1'); // I can suppose it's a signup page  
$I->see('Sign Up', '//body/h1'); // with XPath  
$I->see('Sign Up', ['css' => 'body h1']); // with strict CSS locator

Note that the search is done after stripping all HTML tags from the body, so $I->see('strong') will return true for strings like:

* <p>I am Stronger than thou</p>
* <script>document.createElement('strong');</script>

But will not be true for strings like:

* <strong>Home</strong>
* <div class="strong">Home</strong>
* <!-- strong -->

For checking the raw source code, use seeInSource().

* param string $text
* param string $selector optional

**seeCheckboxIsChecked**

Checks that the specified checkbox is checked.

<?php  
$I->seeCheckboxIsChecked('#agree'); // I suppose user agreed to terms  
$I->seeCheckboxIsChecked('#signup\_form input[type=checkbox]'); // I suppose user agreed to terms, If there is only one checkbox in form.  
$I->seeCheckboxIsChecked('//form/input[@type=checkbox and @name=agree]');  
?>

* param $checkbox

**seeCookie**

Checks that a cookie with the given name is set. You can set additional cookie params like domain, path as array passed in last argument.

<?php  
$I->seeCookie('PHPSESSID');  
?>

* param $cookie
* param array $params

**seeCurrentUrlEquals**

Checks that the current URL is equal to the given string. Unlike seeInCurrentUrl, this only matches the full URL.

<?php  
// to match root url  
$I->seeCurrentUrlEquals('/');  
?>

* param string $uri

**seeCurrentUrlMatches**

Checks that the current URL matches the given regular expression.

<?php  
// to match root url  
$I->seeCurrentUrlMatches('~$/users/(\d+)~');  
?>

* param string $uri

**seeElement**

Checks that the given element exists on the page and is visible. You can also specify expected attributes of this element.

<?php  
$I->seeElement('.error');  
$I->seeElement('//form/input[1]');  
$I->seeElement('input', ['name' => 'login']);  
$I->seeElement('input', ['value' => '123456']);  
  
// strict locator in first arg, attributes in second  
$I->seeElement(['css' => 'form input'], ['name' => 'login']);  
?>

* param $selector
* param array $attributes @return

**seeElementInDOM**

Checks that the given element exists on the page, even it is invisible.

<?php  
$I->seeElementInDOM('//form/input[type=hidden]');  
?>

* param $selector
* param array $attributes

**seeInCurrentUrl**

Checks that current URI contains the given string.

<?php  
// to match: /home/dashboard  
$I->seeInCurrentUrl('home');  
// to match: /users/1  
$I->seeInCurrentUrl('/users/');  
?>

* param string $uri

**seeInField**

Checks that the given input field or textarea equals (i.e. not just contains) the given value. Fields are matched by label text, the “name” attribute, CSS, or XPath.

<?php  
$I->seeInField('Body','Type your comment here');  
$I->seeInField('form textarea[name=body]','Type your comment here');  
$I->seeInField('form input[type=hidden]','hidden\_value');  
$I->seeInField('#searchform input','Search');  
$I->seeInField('//form/\*[@name=search]','Search');  
$I->seeInField(['name' => 'search'], 'Search');  
?>

* param $field
* param $value

**seeInFormFields**

Checks if the array of form parameters (name => value) are set on the form matched with the passed selector.

<?php  
$I->seeInFormFields('form[name=myform]', [  
 'input1' => 'value',  
 'input2' => 'other value',  
]);  
?>

For multi-select elements, or to check values of multiple elements with the same name, an array may be passed:

<?php  
$I->seeInFormFields('.form-class', [  
 'multiselect' => [  
 'value1',  
 'value2',  
 ],  
 'checkbox[]' => [  
 'a checked value',  
 'another checked value',  
 ],  
]);  
?>

Additionally, checkbox values can be checked with a boolean.

<?php  
$I->seeInFormFields('#form-id', [  
 'checkbox1' => true, // passes if checked  
 'checkbox2' => false, // passes if unchecked  
]);  
?>

Pair this with submitForm for quick testing magic.

<?php  
$form = [  
 'field1' => 'value',  
 'field2' => 'another value',  
 'checkbox1' => true,  
 // ...  
];  
$I->submitForm('//form[@id=my-form]', $form, 'submitButton');  
// $I->amOnPage('/path/to/form-page') may be needed  
$I->seeInFormFields('//form[@id=my-form]', $form);  
?>

* param $formSelector
* param $params

**seeInPageSource**

Checks that the page source contains the given string.

<?php  
$I->seeInPageSource('<link rel="apple-touch-icon"');

* param $text

**seeInPopup**

Checks that the active JavaScript popup, as created by window.alert|window.confirm|window.prompt, contains the given string.

* param $text

@throws \Codeception\Exception\ModuleException

**seeInSource**

Checks that the current page contains the given string in its raw source code.

<?php  
$I->seeInSource('<h1>Green eggs &amp; ham</h1>');

* param $raw

**seeInTitle**

Checks that the page title contains the given string.

<?php  
$I->seeInTitle('Blog - Post #1');  
?>

* param $title

**seeLink**

Checks that there’s a link with the specified text. Give a full URL as the second parameter to match links with that exact URL.

<?php  
$I->seeLink('Logout'); // matches <a href="#">Logout</a>  
$I->seeLink('Logout','/logout'); // matches <a href="/logout">Logout</a>  
?>

* param string $text
* param string $url optional

**seeNumberOfElements**

Checks that there are a certain number of elements matched by the given locator on the page.

<?php  
$I->seeNumberOfElements('tr', 10);  
$I->seeNumberOfElements('tr', [0,10]); // between 0 and 10 elements  
?>

* param $selector
* param mixed $expected int or int[]

**seeNumberOfElementsInDOM**

**not documented**

**seeOptionIsSelected**

Checks that the given option is selected.

<?php  
$I->seeOptionIsSelected('#form input[name=payment]', 'Visa');  
?>

* param $selector
* param $optionText

**selectOption**

Selects an option in a select tag or in radio button group.

<?php  
$I->selectOption('form select[name=account]', 'Premium');  
$I->selectOption('form input[name=payment]', 'Monthly');  
$I->selectOption('//form/select[@name=account]', 'Monthly');  
?>

Provide an array for the second argument to select multiple options:

<?php  
$I->selectOption('Which OS do you use?', array('Windows','Linux'));  
?>

Or provide an associative array for the second argument to specifically define which selection method should be used:

<?php  
$I->selectOption('Which OS do you use?', array('text' => 'Windows')); // Only search by text 'Windows'  
$I->selectOption('Which OS do you use?', array('value' => 'windows')); // Only search by value 'windows'  
?>

* param $select
* param $option

**setCookie**

Sets a cookie with the given name and value. You can set additional cookie params like domain, path, expires, secure in array passed as last argument.

<?php  
$I->setCookie('PHPSESSID', 'el4ukv0kqbvoirg7nkp4dncpk3');  
?>

* param $name
* param $val
* param array $params

**submitForm**

Submits the given form on the page, optionally with the given form values. Give the form fields values as an array. Note that hidden fields can’t be accessed.

Skipped fields will be filled by their values from the page. You don’t need to click the ‘Submit’ button afterwards. This command itself triggers the request to form’s action.

You can optionally specify what button’s value to include in the request with the last parameter as an alternative to explicitly setting its value in the second parameter, as button values are not otherwise included in the request.

Examples:

<?php  
$I->submitForm('#login', [  
 'login' => 'davert',  
 'password' => '123456'  
]);  
// or  
$I->submitForm('#login', [  
 'login' => 'davert',  
 'password' => '123456'  
], 'submitButtonName');

For example, given this sample “Sign Up” form:

<form action="/sign\_up">  
 Login:  
 <input type="text" name="user[login]" /><br/>  
 Password:  
 <input type="password" name="user[password]" /><br/>  
 Do you agree to our terms?  
 <input type="checkbox" name="user[agree]" /><br/>  
 Select pricing plan:  
 <select name="plan">  
 <option value="1">Free</option>  
 <option value="2" selected="selected">Paid</option>  
 </select>  
 <input type="submit" name="submitButton" value="Submit" />  
</form>

You could write the following to submit it:

<?php  
$I->submitForm(  
 '#userForm',  
 [  
 'user[login]' => 'Davert',  
 'user[password]' => '123456',  
 'user[agree]' => true  
 ],  
 'submitButton'  
);

Note that “2” will be the submitted value for the “plan” field, as it is the selected option.

Also note that this differs from PhpBrowser, in that

'user' => [ 'login' => 'Davert' ]

is not supported at the moment. Named array keys must be included in the name as above.

Pair this with seeInFormFields for quick testing magic.

<?php  
$form = [  
 'field1' => 'value',  
 'field2' => 'another value',  
 'checkbox1' => true,  
 // ...  
];  
$I->submitForm('//form[@id=my-form]', $form, 'submitButton');  
// $I->amOnPage('/path/to/form-page') may be needed  
$I->seeInFormFields('//form[@id=my-form]', $form);  
?>

Parameter values must be set to arrays for multiple input fields of the same name, or multi-select combo boxes. For checkboxes, either the string value can be used, or boolean values which will be replaced by the checkbox’s value in the DOM.

<?php  
$I->submitForm('#my-form', [  
 'field1' => 'value',  
 'checkbox' => [  
 'value of first checkbox',  
 'value of second checkbox,  
 ],  
 'otherCheckboxes' => [  
 true,  
 false,  
 false  
 ],  
 'multiselect' => [  
 'first option value',  
 'second option value'  
 ]  
]);  
?>

Mixing string and boolean values for a checkbox’s value is not supported and may produce unexpected results.

Field names ending in “[]” must be passed without the trailing square bracket characters, and must contain an array for its value. This allows submitting multiple values with the same name, consider:

$I->submitForm('#my-form', [  
 'field[]' => 'value',  
 'field[]' => 'another value', // 'field[]' is already a defined key  
]);

The solution is to pass an array value:

// this way both values are submitted  
$I->submitForm('#my-form', [  
 'field' => [  
 'value',  
 'another value',  
 ]  
]);

The $button parameter can be either a string, an array or an instance of Facebook\WebDriver\WebDriverBy. When it is a string, the button will be found by its “name” attribute. If $button is an array then it will be treated as a strict selector and a WebDriverBy will be used verbatim.

For example, given the following HTML:

<input type="submit" name="submitButton" value="Submit" />

$button could be any one of the following:

* ‘submitButton’
* [‘name’ => ‘submitButton’]
* WebDriverBy::name(‘submitButton’)
* param $selector
* param $params
* param $button

**switchToIFrame**

Switch to another frame on the page.

Example:

<iframe name="another\_frame" src="http://example.com">

<?php  
# switch to iframe  
$I->switchToIFrame("another\_frame");  
# switch to parent page  
$I->switchToIFrame();

* param string|null $name

**switchToNextTab**

Switches to next browser tab. An offset can be specified.

<?php  
// switch to next tab  
$I->switchToNextTab();  
// switch to 2nd next tab  
$I->switchToNextTab(2);

Can’t be used with PhantomJS

* param int $offset 1

**switchToPreviousTab**

Switches to previous browser tab. An offset can be specified.

<?php  
// switch to previous tab  
$I->switchToPreviousTab();  
// switch to 2nd previous tab  
$I->switchToPreviousTab(2);

Can’t be used with PhantomJS

* param int $offset 1

**switchToWindow**

Switch to another window identified by name.

The window can only be identified by name. If the $name parameter is blank, the parent window will be used.

Example:

<input type="button" value="Open window" onclick="window.open('http://example.com', 'another\_window')">

<?php  
$I->click("Open window");  
# switch to another window  
$I->switchToWindow("another\_window");  
# switch to parent window  
$I->switchToWindow();  
?>

If the window has no name, match it by switching to next active tab using switchToNextTab method.

Or use native Selenium functions to get access to all opened windows:

<?php  
$I->executeInSelenium(function (\Facebook\WebDriver\Remote\RemoteWebDriver $webdriver) {  
 $handles=$webdriver->getWindowHandles();  
 $last\_window = end($handles);  
 $webdriver->switchTo()->window($last\_window);  
});  
?>

* param string|null $name

**typeInPopup**

Enters text into a native JavaScript prompt popup, as created by window.prompt.

* param $keys

@throws \Codeception\Exception\ModuleException

**uncheckOption**

Unticks a checkbox.

<?php  
$I->uncheckOption('#notify');  
?>

* param $option

**unselectOption**

Unselect an option in the given select box.

* param $select
* param $option

**wait**

Wait for $timeout seconds.

* param int|float $timeout secs @throws \Codeception\Exception\TestRuntimeException

**waitForElement**

Waits up to $timeout seconds for an element to appear on the page. If the element doesn’t appear, a timeout exception is thrown.

<?php  
$I->waitForElement('#agree\_button', 30); // secs  
$I->click('#agree\_button');  
?>

* param $element
* param int $timeout seconds @throws \Exception

**waitForElementChange**

Waits up to $timeout seconds for the given element to change. Element “change” is determined by a callback function which is called repeatedly until the return value evaluates to true.

<?php  
use \Facebook\WebDriver\WebDriverElement  
$I->waitForElementChange('#menu', function(WebDriverElement $el) {  
 return $el->isDisplayed();  
}, 100);  
?>

* param $element
* param \Closure $callback
* param int $timeout seconds @throws \Codeception\Exception\ElementNotFound

**waitForElementNotVisible**

Waits up to $timeout seconds for the given element to become invisible. If element stays visible, a timeout exception is thrown.

<?php  
$I->waitForElementNotVisible('#agree\_button', 30); // secs  
?>

* param $element
* param int $timeout seconds @throws \Exception

**waitForElementVisible**

Waits up to $timeout seconds for the given element to be visible on the page. If element doesn’t appear, a timeout exception is thrown.

<?php  
$I->waitForElementVisible('#agree\_button', 30); // secs  
$I->click('#agree\_button');  
?>

* param $element
* param int $timeout seconds @throws \Exception

**waitForJS**

Executes JavaScript and waits up to $timeout seconds for it to return true.

In this example we will wait up to 60 seconds for all jQuery AJAX requests to finish.

<?php  
$I->waitForJS("return $.active == 0;", 60);  
?>

* param string $script
* param int $timeout seconds

**waitForText**

Waits up to $timeout seconds for the given string to appear on the page.

Can also be passed a selector to search in, be as specific as possible when using selectors. waitForText() will only watch the first instance of the matching selector / text provided. If the given text doesn’t appear, a timeout exception is thrown.

<?php  
$I->waitForText('foo', 30); // secs  
$I->waitForText('foo', 30, '.title'); // secs  
?>

* param string $text
* param int $timeout seconds
* param string $selector optional @throws \Exception

Module reference is taken from the source code. [Help us to improve documentation. Edit module reference](https://github.com/Codeception/Codeception/tree/2.3/src/Codeception/Module/WebDriver.php)

Let us help you take Codeception further. We offer a number of paid services that can help you and your organization get the most out of Codeception Enterprise. Software testing & test automation development and outsourcing. **Request a quote for** [**enterprise support**](http://sdclabs.com/codeception?utm_source=codeception.com&utm_medium=docs_bottom&utm_term=link&utm_campaign=reference) **or** [**trainings**](http://sdclabs.com/trainings?utm_source=codeception.com&utm_medium=docs_bottom&utm_term=link&utm_campaign=reference)

Codeception is a BDD-styled PHP testing framework, brought to you by [Codeception Team](http://codeception.com/credits). Logo by [Mr. Adnan](https://twitter.com/adnanblog). OpenSource **MIT Licensed**.

#### Thanks to

* [Installation](http://codeception.com/install)
* [Credits](http://codeception.com/credits)
* [Releases](http://codeception.com/changelog)
* [Enterprise Support](http://sdclabs.com/codeception?utm_source=codeception.com&utm_medium=bottom_menu&utm_term=link&utm_campaign=reference)
* [Trainings](http://sdclabs.com/trainings?utm_source=codeception.com&utm_medium=bottom_menu&utm_term=link&utm_campaign=reference)
* [License](https://github.com/Codeception/Codeception/blob/master/LICENSE)

#### Codeception Family

**[Robo](http://robo.li/)**

Modern PHP **Task Runner**. Allows to declare tasks with zero configuration in pure PHP.

**[CodeceptJS](http://codecept.io/)**

Codeception for **NodeJS**. Write acceptance tests in ES6 and execute in webdriverio, Selenium WebDriver, and Protractor.

**© 2011–2017**

[Follow @Codeception](https://twitter.com/codeception)