2nd Edition

Watir Recipes

The Problem Solving Guide to Watir

Zhimin Zhan

Watir Recipes

The problem solving guide to Watir

Zhimin Zhan

This book is for sale at http://leanpub.com/watir-recipes

This version was published on 2017-02-12

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

© 2013 - 2017 Zhimin Zhan

Also By Zhimin Zhan

Practical Web Test Automation

Selenium WebDriver Recipes in Ruby

Selenium WebDriver Recipes in Java

Learn Ruby Programming by Examples

Learn Swift Programming by Examples

Selenium WebDriver Recipes in Python

API Testing Recipes in Ruby

Selenium WebDriver Recipes in Node.js

To Xindi, for your understanding and support. Thank you!

Contents

Preface . . . . **i**
- Preface to First Edition ii
- Who should read this book iii
- How to read this book iii
- Get recipe test scripts iii
- Send me feedback iv

1. Introduction **1**
- Watir and its variants 1
- RSpec 2
- Run recipe scripts 3

2. Watir and Selenium WebDriver **10**
- Install Selenium Browser Drivers 10
- Cross browser testing with Watir-WebDriver 11
- Selenium WebDriver Locators 12
- Locating elements in Watir 12
- Access underneath Selenium API 13

3. Hyperlink **15**
- Start browser 15
- Click a link by text 15
- Click Nth link with the same link text 17
- Verify a link present or not? 18
- Getting link data attributes 18
- Test links open a new browser window 19

4. Button **20**
- Click a button by text 20

Click a button by ID 21
Click a button by name 21
Click an image button 21
Assert a button present 21
Assert a button displayed or hidden? 22
Assert a button enabled or disabled? 22

Preface

This first edition of this book mainly targets `Watr-Classic` framework, the original `Watir` drives Internet Explorer on Windows platform. In the three years since publication of the first edition, the industry of web application testing continues to evolve:

- Internet Explorer on its way out

  After years of declining use, Internet Explorer is replaced by Microsoft Edge as the default browser on Windows 10. In January 2016, Microsoft ended support for old versions of Internet Explorer, including IE8, IE9 and IE10. While IE won't disappear right away, it is safe to assume that the end of IE is coming.
- Watir team move towards Watir-WebDriver

  The development of Watir-Classic[1], a test framework exclusively for IE, has remained inactive since June 2015, 8 months ago. With the status of Internet Explorer, it is not that surprising. On the contrary, Watir-WebDriver (and now known as Watir 6) receives regular updates from the Watir team.
- Cross-Browser and mobile testing in demand

  Modern web applications target different browsers and platforms, as a result, cross-browser testing and testing on mobile platforms are in high demand. Selenium WebDriver is undisputedly the leader on these, because WebDriver is becoming a W3C standard. Watir-WebDriver combines the intuitive Watir syntax and the power of Selenium WebDriver.

In this second edition, I will use `Watir` as the main framework for the recipes, with revised and 50% new content. But I haven't forgotten `Watir-Classic`, the syntax and features are applicable only to `Watir-Classic` are highlighted.

Zhimin Zhan

Brisbane, Australia

[1] https://github.com/watir/watir-classic

Preface to First Edition

Two years ago I presented at an international conference on software testing. I was really impressed with tester's desire to embrace automate test web applications. The audience surrounded me with various questions after my sessions. The following year, I was invited to present at the same conference. The enthusiastic atmosphere was the same, if not stronger ('Test Automation' and 'Web application testing' were listed as No.1 and No.2 of top 10 hot topics in the audience survey). I did recognize a couple of familiar faces. When I asked them casually how their test automation was going in their projects. They either said "not so good" or shied away. This got me thinking on my trip back.

Not long after, I was coaching test automation on a client site. The only tester there was completely new to test automation, but she was quite keen to learn. At the end of the day, she did well, developed a dozen of automated test cases. I then asked her: "After you read my book or attended one of my presentations, would you go back to try test automation in your project". "No", she quickly answered, "I would believe in it, but I will not have the confidence to give it a try at work." Suddenly, I knew why some of my audience were so keen on test automation but dared not put it into practice at work: lack of confidence.

Everyone in the field understands that manual testing is the bottleneck of software development and performing regression testing is practically impossible for many projects. As a result, long release cycles and poor quality products. Few testers consider manual testing exciting and fun (I can tell you that test automation is). Motivated managers or testers want to change that. However, the knowledge they gain from books or presentations would not give them enough confidence and courage to take action.

Therefore, test automation is rarely done successfully. From executive's perspective, they usually are more cautious after having seen several failed attempts on test automation. Practically, for motivated project managers or test team leaders who plan to introduce test automation, they need to have a secret trial. They usually develop some simple tests by following the start guide. However, they will soon face some challenges such as: clicking this dynamic generated hyperlink, handling base authentication pop ups, …, etc. Often, they got stuck there.

The motivation for me to write this book: to guide these motivated test professionals with writing test scripts in Watir, a popular Ruby testing library for automating web browsers.

This book contains over 110 recipes for testing web applications with Watir. If you have read my other book 'Practical Web Test Automation[2]', you would probably know my style: being

[2] https://leanpub.com/practical-web-test-automation

practical. I will let the test scripts do the most talking. These recipe test scripts are 'live', as I have created web sites and offline web pages for testing. By using this book, sample test scripts and test pages (or sites), you can

1. **Identify** your issue
2. **Find** the recipe
3. **Run** the test case
4. **See** test execution in your browser

Who should read this book

Testers or programmers who write (or want to learn) Watir automated tests to test web applications.

How to read this book

Usually, a 'recipe' book is a reference book. Readers can go directly to the part that interests them. For example, if you are testing a multiple select list and don't know how, you can look up in the Table of Contents, then go to chapter 7. As software testing is so practical, testers can use this book to learn test automation in Watir too, by going through recipes one by one. I have arranged the recipes according to the levels of complexity.

Get recipe test scripts

To help readers learn more effectively, this book has a dedicated site[3] which contains the sample test scripts and related resources. The test scripts are packaged with sample web pages to make it easier and quicker for executing tests.

All recipe test scripts are Watir 5 compliant and run on all major browsers (*watir-classic*: Internet Explorer 10 on Windows 7, *watir*: Firefox, Chrome and IE on Windows, Mac and Linux). I plan to keep the test scripts updated with the latest stable Watir version.

[3]http://zhimin.com/books/watir-recipes

Send me feedback

I would appreciate hearing from you. Comments, suggestions, errors in the book and test scripts are all welcome. You can submit your feedback on the book's site.

Zhimin Zhan

April 2013

1. Introduction

Watir (Web Application Testing in Ruby) is a free and open source library for automated testing web applications in web browsers. I assume you already know something about Watir, simply based on the fact that you have picked up this book or opened it in your eBook reader.

Watir and its variants

The '**r**' in Watir stands for 'Ruby', a free and powerful dynamic language with concise and elegant syntax. In other words, Watir test scripts are Ruby scripts. Inspired by Watir's success, there are clone frameworks in .NET and Java platforms: WatiN and Watij respectively. In my view, these two test frameworks are of not much value. There is a clear reason why creators of Watir included Ruby in the name, simply because of its importance (popular web framework Ruby on Rails also includes Ruby in its name). The concise, intuitiveness and flexibility of Ruby programming language makes it an ideal choice for automated test scripts. By the way, Ruby is a scripting language, Java and C# are not.

The current Watir version 6 (released in November 2016) was known as Watir-WebDriver. As its name suggests, WebDriver-backed Watir, i.e. Watir syntax test scripts with Selenium-WebDriver as the engine underneath.

Watir-Classic

Watir Classic is a Watir driver for automating Internet Explorer on Windows. Watir-classic directly drives the browser through Microsoft's OLE protocol. The original Watir supports IE only, later, the *watir* gem was renamed to *watir-classic*. The *watir* gem became an umbrella gem contains *watir-classic* and *watir-webdriver* (until November 2016).

The below is a simple Watir-Classic test script to open a new IE browser window and navigate to the Watir home page.

```
require 'watir-classic'
browser = Watir::Browser.new
browser.goto("http://watir.com")
```

In this book, I will focus on Watir (6.0+), except a few recipes that are applicable only to `Watir-Classic` (and they are highlighted).

RSpec

Watir drives browsers, however, to make the effective use of Watir scripts for testing, we need to put them in a test framework which defines test structures and provides assertions (to perform checks in test scripts). Typical choices are:

- xUnit Unit Test Frameworks such as JUnit, NUnit.
- Behaviour Driven Frameworks such as RSpec, Cucumber.

In this book, I mainly use RSpec, the de facto Behaviour Driven Development (BDD) framework for Ruby.

```
describe "A grouped collection of test case " do
  before(:all) do
    @browser = Watir::Browser.new
  end

  after(:all) do
    browser.close unless debugging?
  end

  before(:each) do
    browser.goto(site_url)
  end

  # 'it' marks the start of a test case, ends with the matching 'end'
  it "Check page title" do
    expect(browser.title).to eq("Watir Recipes")
    expect(browser.title.include?("Watir")).to be_truthy
```

```
    expect(browser.title.include?("Watir")).to be_falsey
    expect(browser.title).not_to include("Selenium")
  end

  # more test cases ...

end
```

The keywords `describe`, `before`, `after` and `it` define the structure of a test script.

- **`describe "..."` do**

  Description of a collection of related test cases
- **`before()`** and **`after()`**.

  Optional test statements run before and after each or all test cases.
- **`it "..." do`**

  Individual test case.

`expect(...).to` statements are called rspec-expectations, which are used to perform checks. There is also an older `should`-based syntax, which is still supported in RSpec but deprecated. Here is the `should-syntax` version of the above example:

```
browser.title.should == "Watir Recipes"
browser.title.include?("Watir").should be_truthy
browser.title.include?("Watir").should_not be_falsey
browser.title.should_not include("Selenium")
```

You will find more about RSpec from its home page[1]. However, I honestly don't think it is necessary. The part used for test scripts is not much and quite intuitive. After studying and trying out some examples, you will be quite comfortable with RSpec.

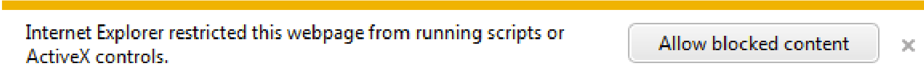
Run recipe scripts

Test scripts for all recipes can be downloaded from the book's site. They are all in ready-to-run state. I include the target web pages or sites as well as Watir test scripts. There are two kinds of target web pages: local HTML files and web pages on a live site.

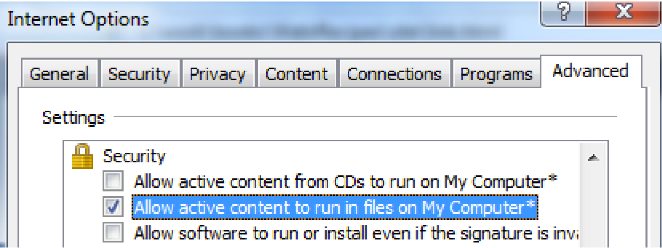
[1] http://rspec.info

Enable IE to allow executing JavaScript locally

Running tests against local HTML files is fast and reliable (no need for Internet Access), and readers can add or modify web pages to try out new test operations. However, when opening local web pages containing JavaScript, you may see the security warning "Internet Explorer restricted this web page from running scripts or ActiveX controls".



This prevents the execution of JavaScripts from the local files. As a result, it affects the test execution. To disable this warning (to be sure, you may want to inspect the only jquery.js and inline javascript in the sample HTML files), open 'Internet Options' in Internet Explorer. Go to 'Advanced' tab, scroll down to the security section there should be an option "Allow active content to run in files on My Computer". Enabling this should allow them to run.

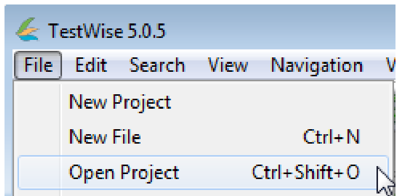


Run tests in TestWise

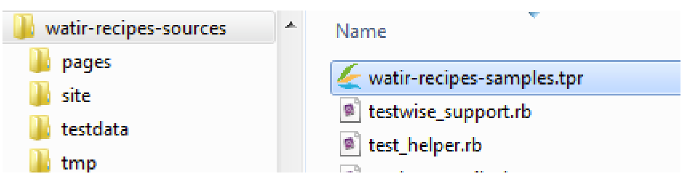
In this book, I refer to TestWise 5, a functional testing Integration Development Environment (IDE) that supports Watir and Selenium WebDriver, when editing or executing test scripts. If you have a preferred testing IDEs such as Apatana Studio and NetBeans 6 or code editors such as Sublime Text and TextMate, go for it. It shall not affect your learning this book or running recipe test scripts.

Installation of TestWise is easy. It only takes a couple of minutes (unless your Internet speed is very slow) to download and install. TestWise is the only software you need to use while learning with this book or developing Watir test scripts for your work.

To open recipe test scripts, close currently opened project (if there is one). Select menu File → Open Project,



Select the project file *watir-recipes-sources\watir-recipes-samples.tpr*

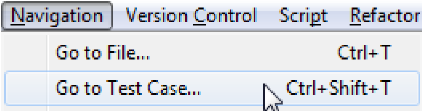


The TestWise window shall appear as below:

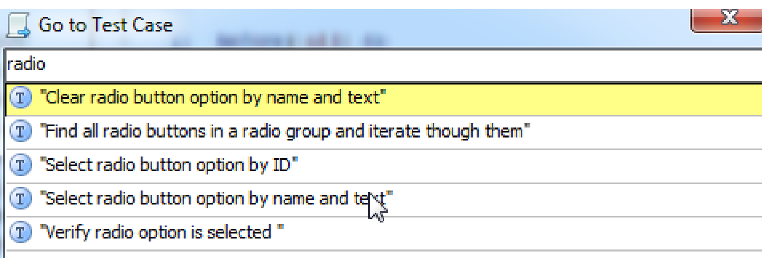
Find the test case

You can locate the recipe either by following the chapters or searching by names. There are over 150 test cases in one test project. Here is the quickest way to find the one you want in TestWise.

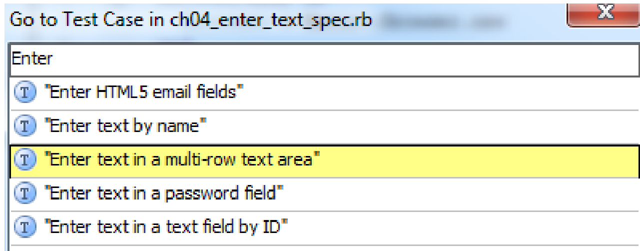
Select the menu 'Navigation' → 'Go to Test Case..'.



The "Go to Test Case" window appears with a list of the test cases from the project. The searching starts as you type.



If you want to find a test case within a test script file, in the editor, press Ctrl+F12 to show and select the test cases within the "Go to Test Case" window.

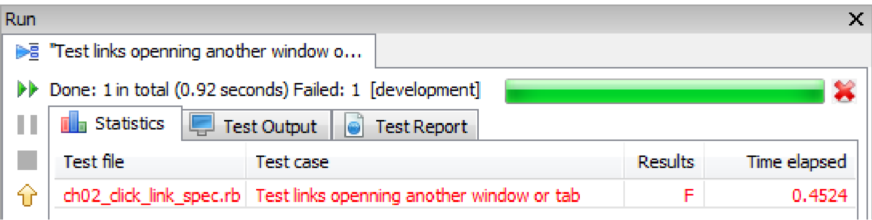


Run individual test case

Move the cursor to a line within a test case (between `it "..." do` and `end`). Right click and select "Run '…'".

```
it "Verify checkbox is selected" do
  browser.checkbox(:name => "vehicle_bike").set
  expect(browser.checkbox(:name => "vehicle_bike").set?).to be_truthy
  browser.checkbox(:name => "vehicle_bike").click
  expect(browser.checkbox(:name => "vehicle_bike").set?).to be_falsey
      Run "Verify checkbox is selected"          Ctrl+Shift+F10
end
```

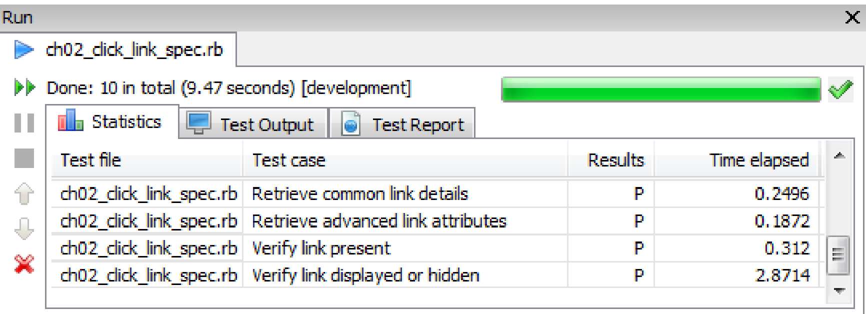
The below is a screenshot of execution panel when one test case failed,



Run all test cases in a test script file

You can also run all test cases in currently opened test script file by clicking the blue triangle button on the top toolbar.

The below is a screenshot of the execution panel when all test cases in a test script file passed.



Run tests from command line

One advantage of open-source test frameworks, such as Watir and Selenium, is FREEDOM. You can edit the test scripts in any text editor and run them from a command line.

You will need to install Ruby first, then install RSpec and the preferred web test driver and library (known as Gem in Ruby). Basic steps are:

- install Ruby interpreter

  Window installer: http://rubyinstaller.org Linux or Mac: included or compile from source
- install RSpec

  > gem install rspec
- install test framework gem(s)

  > gem install watir

  or

  > gem install watir-classic

For windows users, you may simply download and install the free pre-packaged RubyShell (based on Ruby Windows Installer) at *http://testwisely.com/testwise/downloads*[2].

Once the installation is complete (it takes about 1 minute), we can run an RSpec test from the command line. You need to have some knowledge of typing commands in a console (Unix) or command prompt.

To run test cases in a test script file (e.g. google_spec.rb), enter the command

```
> rspec google_spec.rb
```

NOTE: remember to navigate or change the directory to the files' location.

Run multiple test script files in one go:

```
> rspec first_spec.rb  second_spec.rb
```

Run individual test case in a test script file, supply a line number in chosen test case range.

```
> rspec google_spec.rb:30
```

To generate a test report in HTML (under the current directory) after test execution:

[2] http://testwisely.com/testwise/downloads

```
> rspec -fh google_spec.rb > test_report.html
```

The command syntax is the same for Mac OS X and Linux platforms.

2. Watir and Selenium WebDriver

Watir is built on Selenium WebDriver, in other words, Watir syntax on top of Selenium. Selenium WebDriver, aka Selenium 2, is another free and open-source web test automation library. Many like the simple and elegant syntax of Watir, but want to use Selenium WebDriver's strength on multi-browser support.

Install Selenium Browser Drivers

Different from `watir-classic` (drives only Internet Explorer on Windows platform), Watir can drive tests in all major browsers: Chrome, Firefox, IE and Edge on Windows, Mac and Linux platforms. Besides the browser itself, its corresponding driver server also needs to installed (except for Firefox).

- ChromeDriver

  Download ChromeDriver[1] for your platform (Windows, Mac or Linux) and include `chromedriver` location in your PATH environment variable.

- GeckoDriver for Firefox

  Download geckodriver[2] for Firefox `v47+` and include `geckodriver` in your PATH. *Firefox up to v46 comes with WebDriver support.*

- IE Driver Server

  Download IE Driver Server[3] (choose 32bit or 64bit version based on your OS) and place it in a directory in PATH. Configuration is required for your IE browser depending its version, see IE and IEDriverServer Runtime Configuration[4] for details.

- Microsoft WebDriver for Edge

  Download and install Microsoft WebDriver[5]

[1] https://sites.google.com/a/chromium.org/chromedriver/downloads
[2] https://github.com/mozilla/geckodriver/releases/
[3] http://www.seleniumhq.org/download/
[4] https://code.google.com/p/selenium/wiki/InternetExplorerDriver#Required_Configuration
[5] https://developer.microsoft.com/en-us/microsoft-edge/tools/webdriver/

Cross browser testing with Watir-WebDriver

```
require 'watir'

describe "Same test on 4 different browsers" do

  it "Watir IE" do
    browser = Watir::Browser.new(:ie)
    browser.goto("http://testwisely.com/demo")
    browser.link(:text, "NetBank").click
    browser.text_field(:amount, "299" )
    browser.quit
  end

  it "Watir Firefox" do
    browser = Watir::Browser.new(:firefox)
    browser.goto("http://testwisely.com/demo")
    browser.link(:text, "NetBank").click
    browser.text_field(:amount, "299" )
    browser.quit
  end

  it "Watir Chrome" do
    browser = Watir::Browser.new(:chrome)
    browser.goto("http://testwisely.com/demo")
    browser.link(:text, "NetBank").click
    browser.text_field(:amount, "299" )
    browser.quit
  end

  it "Watir Edge" do
    browser = Watir::Browser.new(:edge)
    browser.goto("http://testwisely.com/demo")
    browser.link(:text, "NetBank").click
    browser.text_field(:amount, "299" )
    browser.quit
  end
```

```
end
```

Please note that the execution of IE in Watir-WebDriver is different from Watir-Classic, which is based on OLE.

Selenium WebDriver Locators

As you might have already figured out, to drive an element in a page, we need to find it first. Selenium WebDriver uses what is called locators to find and match the elements on web page. There are 8 locators in Selenium:

Locator	Example
ID	`find_element(:id, "user")`
Name	`find_element(:name, "username")`
Link Text	`find_element(:link_text, "Login")`
	`find_element(:link, "Login")`
Partial Link Text	`find_element(:partial_link_text, "Next")`
XPath	`find_element(:xpath, "//div[@id="login"]/input")`
Tag Name	`find_element(:tag_name, "body")`
Class Name	`find_element(:class_name, "table")`
	`find_element(:class, "body")`
CSS	`find_element(:css, "#login > input[type="text"]")`

You may use any one of them to narrow down the element you are looking for.

Here is a sample Selenium-WebDriver test script.

```
driver = Selenium::WebDriver.for(:chrome)
driver.navigate.to("http://travel.agileway.net")
driver.find_element(:id, "username").send_keys("agileway")
driver.find_element(:name, "password").send_keys("testwise")
driver.find_element(:xpath,"//input[@value='Sign in']").click
```

Locating elements in Watir

Different from Selenium's generic approach of using `find_element` to locate a control on a web page, Watir syntax is based control types.

```
browser = Watir::Browser.new(:chrome)
browser.goto("http://travel.agileway.net")
browser.text_field(:id, "username").set("agileway")
browser.text_field(:name, "password").set(password)
browser.button(:value,"Sign in").click
```

You can use the same way for most of HTML controls, including display only tags such `label` and `span`. The full list is available on docs[6].

What about non-standard tag?

If the control you refer to is not in the above tag list or simply you don't care the tag, you may just use `element`.

```
browser.element(:id, "not_standard_tag")
```

Convert Watir Element to Selenium Element

```
watir_elem = browser.text_field(:id, "username")
selenium_elem = watir_elem.wd
```

By converting to a Selenium element, you can use its functions[7].

Access underneath Selenium API

To master Watir-WebDriver, in my opinion, it is necessary to understand the underneath library: Selenium WebDriver. We can use Selenium API directly using `driver` like the example below:

[6] http://watir.github.io/watir-webdriver/doc/Watir/Container.html
[7] https://github.com/SeleniumHQ/selenium/blob/master/rb/lib/selenium/webdriver/common/element.rb

```
browser = Watir::Browser.new(:chrome)
browser.goto("http://testwisely.com/demo")
browser.driver.manage().window().resize_to(1024, 768) # Selenium
browser.driver.find_element(:link_text, "NetBank").click
browser.select_list(:name, "account").select("Savings") # Watir
browser.driver.find_element(:name, "amount").send_keys("299")
browser.button(:value, "Transfer").click
browser.close
```

For more Selenium examples, you may check out Selenium WebDriver Recipes in Ruby[8].

[8] https://leanpub.com/selenium-recipes-in-ruby

3. Hyperlink

Hyperlinks (or links) are fundamental elements of web pages. As a matter of fact, it is hyperlinks that makes the World Wide Web possible. A sample link is provided below, along with the HTML source.

Recommend Watir

HTML Source

```
<a href="index.html" id="like_watir_link" class="nav" data-id="123" style="f\
ont-size: 14px;">Like Watir</a>
```

Start browser

Testing a website starts with opening a browser.

```
browser = Watir::Browser.new
browser.goto("http://testwisely.com/testwise")
```

Watir-Classic

Watir-Classic may use `browser.start` as well.

```
browser = Watir::Browser.new
browser.start("http://testwisely.com/demo")
```

Click a link by text

Using text is probably the most direct way to click a link in Watir, as it is what we see on the page.

```
browser.link(:text, "Like Watir").click
```

Click a link by ID

Using IDs is the easiest and the safest way to locate an element in HTML. If the page is W3C HTML conformed[1], the IDs should be unique and identified in web controls. In comparison to texts, test scripts that use IDs are less prone to application changes (e.g. an application designer or developer may decide to change the label, but less likely to change the ID).

```
browser.link(:id, "sign_in_link").click
```

Furthermore, if you are testing a web site with multiple languages, using IDs is probably the only feasible option. You do not want to write test scripts like below:

```
if is_chinese?  # a helper function determines the locale
  browser.link(:text, "□□").click
elsif is_italian?
  browser.link(:text, "Accedi").click
else
  browser.link(:text, "Sign in").click
end
```

Click a link by partial text

Watir allows you to identify a hyperlink control with a partial text. This can be quite useful when the text is dynamically generated. In other words, the text on one web page might be different on your next visit. We might be able to use the common text shared by these dynamically generated link texts to identify them.

```
browser.link(:text, /partial/i).click  # contains 'partial', case insensitive
expect(browser.text).to include("This is partial link page")

# alternate should-syntax, however, not recommended
browser.text.should include("This is partial link page")
```

[1] http://www.w3.org/TR/WCAG20-TECHS/H93.html

Here we use the regular expression in ("/…/"), a powerful pattern matching language. If you are not familiar with the regular expression, don't feel intimidated. The use of regular expression in automated test scripts is very minimal. Online regular expression testers such as Rubular[2] will make it easy to learn what you need.

By URL

```
browser.link(:url, "http://testwisely.com/demo").click
```

Please note `:url` only works for Watir Classic (not Watir).

Click Nth link with the same link text

It is not uncommon that there are more than one link with exactly the same text. By default, Watir will choose the first one. What if you want to click the second or Nth one?

The web page below contains three 'Show Answer" links,

1. Do you think automated testing is important and valuable? Show Answer
2. Why didn't you do automated testing in your projects previously? Show Answer
3. Your project now has so comprehensive automated test suite, What changed? Show Answer

To click the second one,

```
browser.link(:text => "Show Answer", :index => 1).click
```

The `:index` tells Watir which element to select in appearing order, Watir (since version 2.0) uses 0-based indexing, i.e. the first one is 0.

```
browser.link(:text, "Show Answer").click  # this will click first link
browser.link(:text => "Show Answer", :index => 0).click # still first link
```

If there are multiple links with the same text that have different attribute values, such as 'class', we could use the attribute to narrow down the choice, such as

[2] http://rubular.com/

```
browser.link(:text => "Same link", :class => "small").click
```

Verify a link present or not?

```
expect(browser.link(:text, "Sign in").present?).to be_truthy
expect(browser.link(:id, "sign_out_link").present?).not_to be_truthy
```

Besides `present?`, you may use `visible?` to check whether an element is visible on the page.

```
browser.link(:text, "Hide").click
expect(browser.link(:text, "Like Watir").present?).to be_falsey
expect(browser.link(:text, "Like Watir").visible?).to be_falsey
```

Watir-Classic

Watir-Classic also has another similar method `exists?`.

Getting link data attributes

Once a control is identified, we can get its other attributes of the element. This is generally applicable to most of the controls.

```
expect(browser.link(:text, "Like Watir").href).to eq(site_url.gsub("link.htm\
l", "index.html"))
expect(browser.link(:text, "Like Watir").id).to eq("like_watir_link")
expect(browser.link(:id, "like_watir_link").text).to eq("Like Watir")
expect(browser.link(:id, "like_watir_link").tag_name).to eq("a")
```

Also you can get the value of custom attributes of this element,

```
expect(browser.link(:id, "like_watir_link").attribute_value("data-id")).to e\
q("123")
```

and its inline CSS style.

```
expect(browser.link(:id, "like_watir_link").attribute_value("style")).to eq(\
"font-size: 14px;")
```

Watir-Classic

A note on the 'style' attribute: the syntax is different between Watir-Classic and Watir-WebDriver.

```
expect(browser.link(:id, "like_watir_link").style).to eq("font-size: 14px;")
# Please note using attribute_value("style") won't work in Watir-Classic
```

Test links open a new browser window

Clicking the link below will open the linked URL in a new browser window or tab.

```
<a href="http://testwisely.com/demo" target="_blank">Open new window</a>
```

For Watir-Classic, we could use `attach` method (see chapter 10) to find the new browser/tab window, it will be easier to perform all testing within one browser window. Here is how:

```
current_url = browser.url
new_window_url = browser.link(:text, "Open new window").href
browser.goto(new_window_url)
# ... testing on new site
browser.text_field(:name, "name").set "sometext"
browser.goto(current_url) # back
```

In this test script, we use a local variable (a programming term) 'current_url' to store the current URL.

4. Button

Buttons can come in two forms - standard and submit buttons. Standard buttons are usually created by the 'button' tag, whereas submit buttons are created by the 'input' tag (normally within form controls).

Standard button

Choose Watir

Submit button in a form

Username: Submit

HTML Source

```
<button id="choose_watir_btn" class="nav" data-id="123" style="font-size: 14\
px;">Choose Watir</button>
<!-- ... -->
<form name="input" action="index.html" method="get">
  Username: <input type="text" name="user">
  <input type="submit" name="submit_action" value="Submit">
</form>
```

Please note that some buttons are actually not buttons, but are hyperlinks styled by CSS.

Click a button by text

```
browser.button(:value, "Choose Watir").click
```

For an input button (in a HTML input tag) in a form, the text shown on the button is the 'value' attribute which might contain extra spaces or invisible characters. Watir expects an exact match when searching an input control by value.

```
<input type="submit" name="submit_action" value="Space After "/>
```

```
# the below will fail
# browser.button(:value, "Space After").click
browser.button(:value, "Space After ").click
```

Click a button by ID

As always, a better way to identify a button is to use IDs. This applies to all controls.

```
browser.button(:id, "choose_watir_btn").click
```

Click a button by name

For an input button, we can use a new generic attribute `name` to locate the control.

```
browser.button(:name, "choose_watir").click
```

Click an image button

There is also another type of 'button': an image that works as a submit button in a form.

```
<input type="image" src="images/go.gif">
```

Besides using ID, the button can be identified by using :src attribute.

```
browser.button(:src, /go/).click
```

`/go/` is a regular expression, it means to locate a button whose `src` attribute contains 'go'.

Assert a button present

Just like hyperlinks, we can use `present?` to check whether a control is present on a web page. This check applies to most of the web controls in Watir.

```
expect(browser.button(:text, "Choose Watir").present?).to be_truthy
expect(browser.button(:text, "Choose Selenium").present?).not_to be_truthy
expect(browser.button(:id, "choose_watir_btn").present?).to be_truthy
```

Assert a button displayed or hidden?

```
expect(browser.button(:id, "choose_watir_btn").visible?).to be_truthy
expect(browser.button(:text, "Choose Watir").visible?).to be_falsey
```

Assert a button enabled or disabled?

A web control can be in a disabled state. A disabled button is un-clickable, and it is displayed differently.

Choose Watir

Normally enabling or disabling buttons (or other web controls) are triggered by JavaScript.

```
expect(browser.button(:text, "Choose Watir").enabled?).to be_truthy
browser.link(:text, "Disable").click
sleep 0.5
expect(browser.button(:id, "choose_watir_btn").enabled?).to be_falsey
browser.link(:text, "Enable").click
sleep 1
expect(browser.button(:id, "choose_watir_btn").enabled?).to be_truthy
```