watir

tutorialspoint

SIMPLY EASY LEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia

https://twitter.com/tutorialspoint

About the Tutorial

Watir (Web Application Testing in Ruby), pronounced as "Water" is an open source tool developed using Ruby which helps in automating web application no matter which language the application is written. The browsers supported are Internet Explorer, Firefox, Chrome, Safari, and Edge. Watir is available as Rubygems gem for installation.

Audience

This tutorial is designed for software programmers who want to learn the basics of Watir and learn to automate browser side testing. This tutorial will give you in detail about the working of Watir with browsers like Internet Explorer, Firefox, Chrome, Safari, and Edge.

Prerequisites

This tutorial is written assuming that the learner has a basic understanding of Ruby.

If you are new to Ruby, we suggest that you pick up a Ruby tutorial first before starting your journey with Watir.

Copyright & Disclaimer

Table of Contents

1. Watir — Overview

Watir (Web Application Testing in Ruby), pronounced as "Water" is an open source tool developed using Ruby which helps in automating web application that suits applications developed in any programming language. The browsers supported for Watir installation Internet Explorer, Firefox, Chrome, Safari, and Edge. Watir is available as Rubygems for installation.

Watir will connect to the browser given and follows up the instructions of opening the URL, clicking on the button, entering data inside a textbox just as any real person will do. Watir is most commonly used for its browser support with testing frameworks like RSpec, Cucumber, etc.

Features of Watir

Watir is rich in features, as discussed below:

Location web elements: There are different ways you can locate web-elements rendered inside the browser. The ones mostly used are id, class, tag name, custom attributes, label etc.

Taking Screenshots: Watir allows you to take screenshot of the testing done as and when required. This helps to keep track of the intermediate testing.

Page Performance: You can easily measure page performance using the performance object which has properties like, *performance.navigation*, *performance.timing*, *performance.memory* and *performance.timeOrigin*. These details are obtained when you connect to the browser.

Page Objects: Page object in Watir will help us to reuse the code in the form of classes. Using this feature, we can automate our app without having to duplicate any code and also make it manageable.

Downloads: With Watir, it is easy to test file download for UI or website.

Alerts: Watir provides easy to use APIs to test alerts popup in your UI or website.

Headless Testing: Using headless testing, the details are obtained in the command line without having to open the browser. This helps to execute UI test cases at the command line.

Advantages of Using Watir

Watir offers the following advantages:

- Watir is an open source tool and very easy to use.
- Watir is developed in Ruby and any web application that works in a browser can be easily automated using watir.
- All the latest browsers are supported in Watir making it easy for testing.

- Watir has inbuilt libraries to test page-performance, alerts, iframes test, browser windows, take screenshots etc.

Disadvantages of Watir

Like any other software, Watir also has its limitations:

- Watir is supported only for Ruby test framework and it cannot be used with any other testing frameworks.

- Mobile testing using Watir is not enhanced and desktop browsers are mimicked to behave like mobile browsers instead of acting as real time devices.

2. Watir — Introduction

Watir (Web Application Testing in Ruby) pronounced as "Water" is an open source tool developed using Ruby which helps in automating web application no matter which language the application is written. Watir comes with a rich set of APIs which helps us interact with the browser, locate page elements, take screenshots, work with alerts, file downloads, *window.open* popup windows, headless testing, etc.

The browsers supported are:

- Internet Explorer
- Firefox
- Chrome
- Safari
- Edge

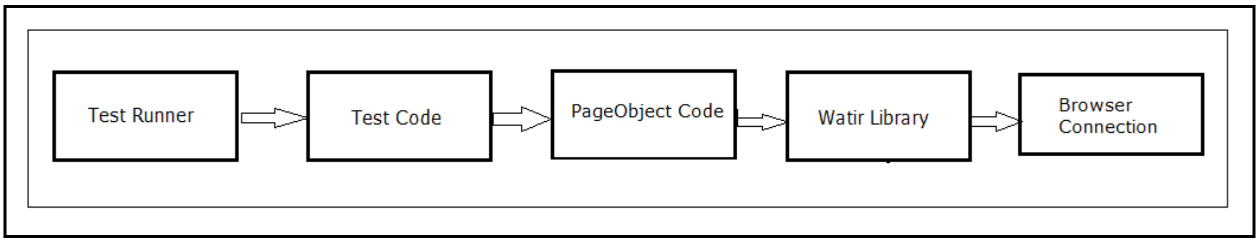
Note: Watir is available as Rubygems gem for installation.

Using Watir webdriver, you can test your websites and UI applications. As Watir fully focuses on the browser related stuff, you can use Watir along with other test framework such as:

- RSpec
- Cucumber

The main testing code will be written using the test framework and interacting with the browser will be done with the help of Watir.

The flow of test framework along with Watir is as shown below:



RSpec or Cucumber are involved in test runner and test code. The details about the website or UI testing can be broken down into page object which will have reference to Watir, wherein it will get the page locators to be used for testing. Watir, along with its webdriver, helps in connecting to the browser and carry out the test automation.

3. Watir — Environment Setup

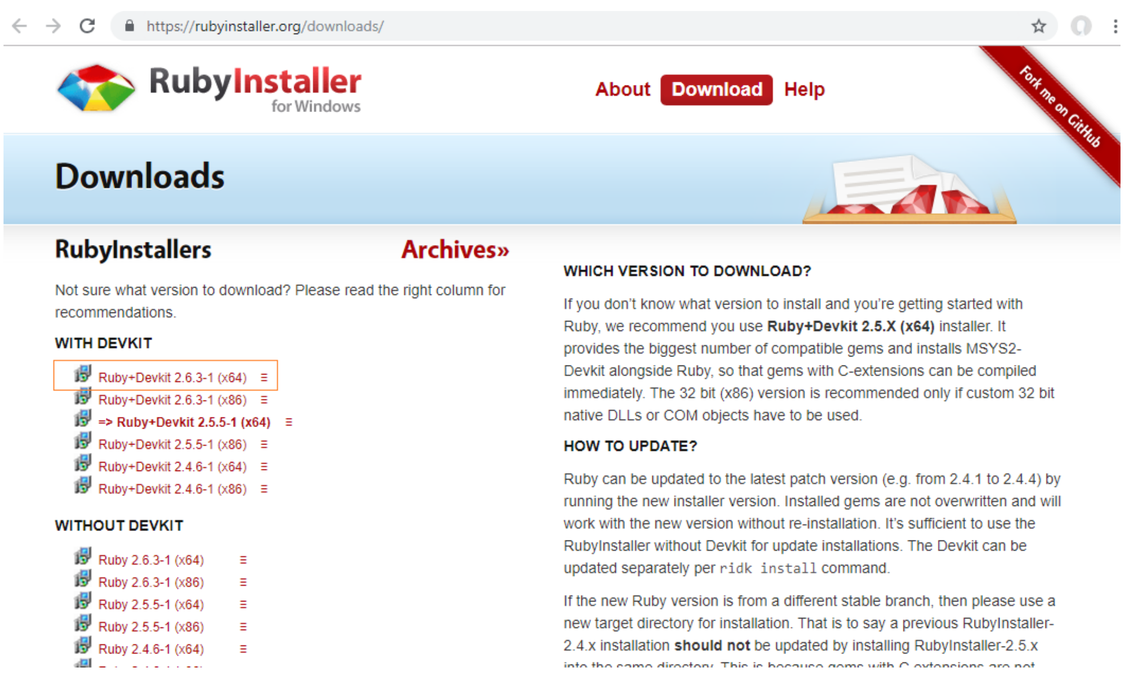
To work with Watir, we need to install the following:

- Install Ruby
- Install Watir
- Ruby Mine (IDE)

Since Watir is build using Ruby, we need to install Ruby and gem (package manager for Ruby).

Ruby Installation on Windows

To install Ruby on Windows, go to : https://rubyinstaller.org/downloads/


https://rubyinstaller.org/downloads/
RubyInstaller for Windows
About
Download
Help
Fork me on GitHub

Downloads

RubyInstallers

Archives»

Not sure what version to download? Please read the right column for recommendations.

WITH DEVKIT

Ruby+Devkit 2.6.3-1 (x64)
Ruby+Devkit 2.6.3-1 (x86)
=> Ruby+Devkit 2.5.5-1 (x64)
Ruby+Devkit 2.5.5-1 (x86)
Ruby+Devkit 2.4.6-1 (x64)
Ruby+Devkit 2.4.6-1 (x86)

WITHOUT DEVKIT

Ruby 2.6.3-1 (x64)
Ruby 2.6.3-1 (x86)
Ruby 2.5.5-1 (x64)
Ruby 2.5.5-1 (x86)
Ruby 2.4.6-1 (x64)

WHICH VERSION TO DOWNLOAD?

If you don't know what version to install and you're getting started with Ruby, we recommend you use **Ruby+Devkit 2.5.X (x64)** installer. It provides the biggest number of compatible gems and installs MSYS2-Devkit alongside Ruby, so that gems with C-extensions can be compiled immediately. The 32 bit (x86) version is recommended only if custom 32 bit native DLLs or COM objects have to be used.

HOW TO UPDATE?

Ruby can be updated to the latest patch version (e.g. from 2.4.1 to 2.4.4) by running the new installer version. Installed gems are not overwritten and will work with the new version without re-installation. It's sufficient to use the RubyInstaller without Devkit for update installations. The Devkit can be updated separately per `ridk install` command.

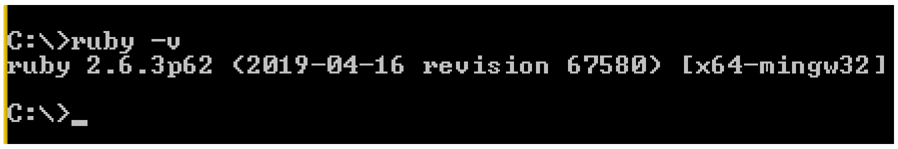
If the new Ruby version is from a different stable branch, then please use a new target directory for installation. That is to say a previous RubyInstaller-2.4.x installation **should not** be updated by installing RubyInstaller-2.5.x

Install the ruby version based on your 32 or 64 bit operating system. Here we have installed the highlighted one as shown in the screenshot. Once you download it, follow the steps as prompted and install accordingly.

When you are done with the installation, check if ruby is installed by checking the version in command line as shown below:

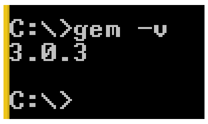
```
C:\> ruby -v
```

If Ruby is successfully installed, you can see an output as shown below:



```
C:\>ruby -v
ruby 2.6.3p62 (2019-04-16 revision 67580) [x64-mingw32]

C:\>_
```

The version installed is 2.6.3. We have successfully installed Ruby on our system. Along with Ruby installation gem i.e. ruby package manager is also installed. You can check the version of gem installed as follows:



```
C:\>gem -v
3.0.3

C:\>
```

We are done with the installation of Ruby and Gem on windows.

Ruby Installation on Linux

For installing Ruby on Linux, run the following commands in your Linux terminal:

```
wget -O ruby-install-0.6.0.tar.gz https://github.com/postmodern/ruby-
install/archive/v0.6.0.tar.gz

tar -xzvf ruby-install-0.6.0.tar.gz

cd ruby-install-0.6.0/

sudo make install

ruby-install ruby 2.5.3
```

Ruby Installation for Mac

For installing Ruby on Linux, run below commands in your Mac terminal:

Install xcode

```
xcode-select --install
```

Install HomeBrew

```
/usr/bin/ruby -e "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

Install rbenv

```
brew install rbenv

rbenv init

touch ~/.bash_profile

echo 'eval "$(rbenv init -)"' >> ~/.bash_profile

source ~/.bash_profile
```

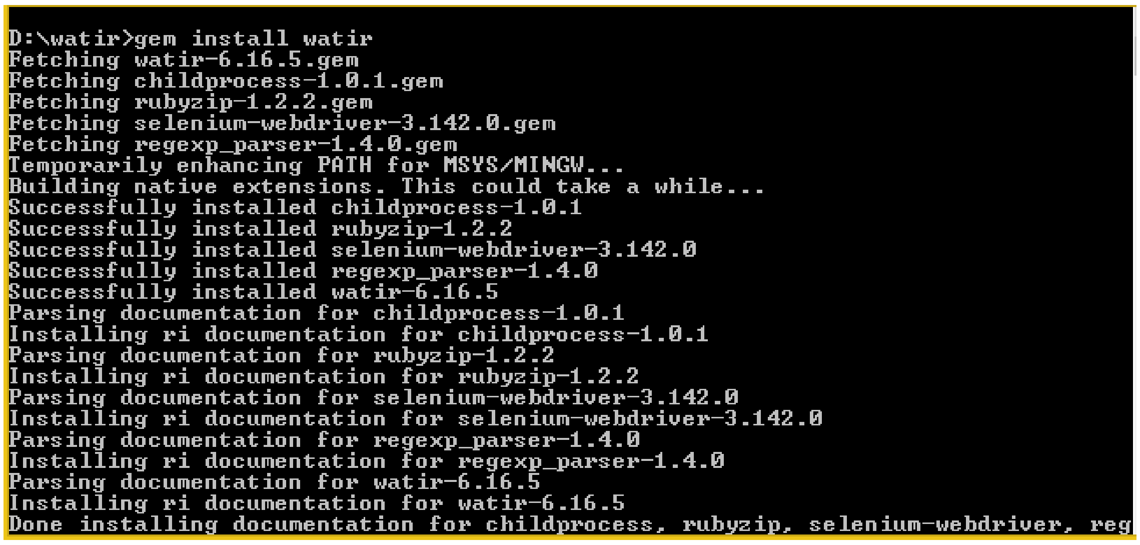
```
rbenv install 2.5.3

rbenv local 2.5.3
```

Installing Watir

The command to install watir is:

```
gem install watir
```

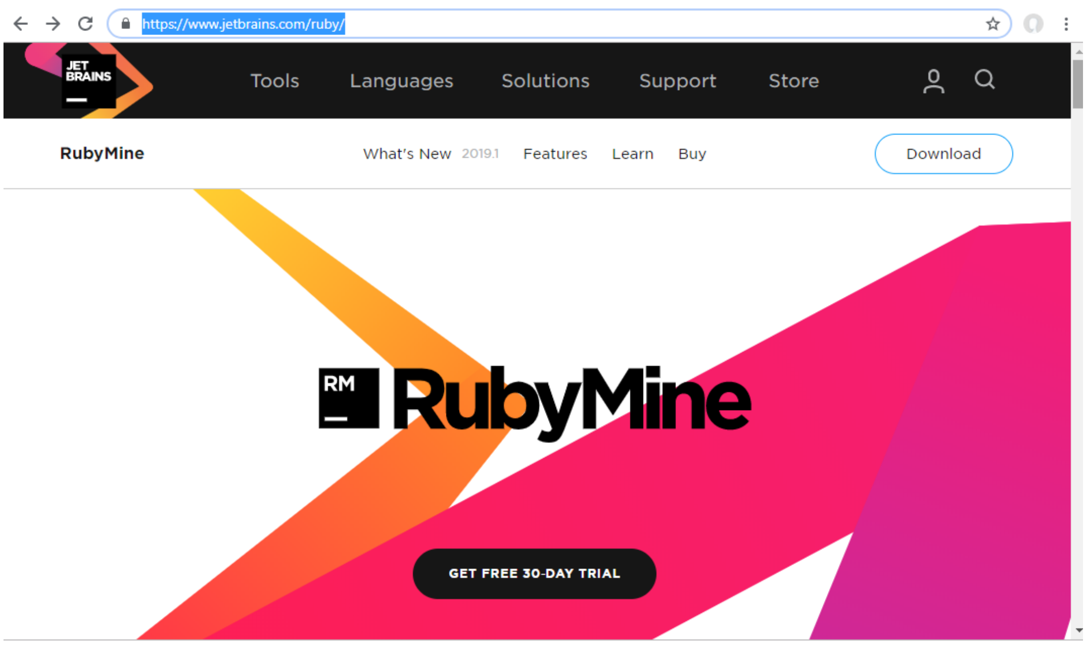
Then you can observe an output screen as shown below:



```
D:\watir>gem install watir
Fetching watir-6.16.5.gem
Fetching childprocess-1.0.1.gem
Fetching rubyzip-1.2.2.gem
Fetching selenium-webdriver-3.142.0.gem
Fetching regexp_parser-1.4.0.gem
Temporarily enhancing PATH for MSYS/MINGW...
Building native extensions. This could take a while...
Successfully installed childprocess-1.0.1
Successfully installed rubyzip-1.2.2
Successfully installed selenium-webdriver-3.142.0
Successfully installed regexp_parser-1.4.0
Successfully installed watir-6.16.5
Parsing documentation for childprocess-1.0.1
Installing ri documentation for childprocess-1.0.1
Parsing documentation for rubyzip-1.2.2
Installing ri documentation for rubyzip-1.2.2
Parsing documentation for selenium-webdriver-3.142.0
Installing ri documentation for selenium-webdriver-3.142.0
Parsing documentation for regexp_parser-1.4.0
Installing ri documentation for regexp_parser-1.4.0
Parsing documentation for watir-6.16.5
Installing ri documentation for watir-6.16.5
Done installing documentation for childprocess, rubyzip, selenium-webdriver, reg
```

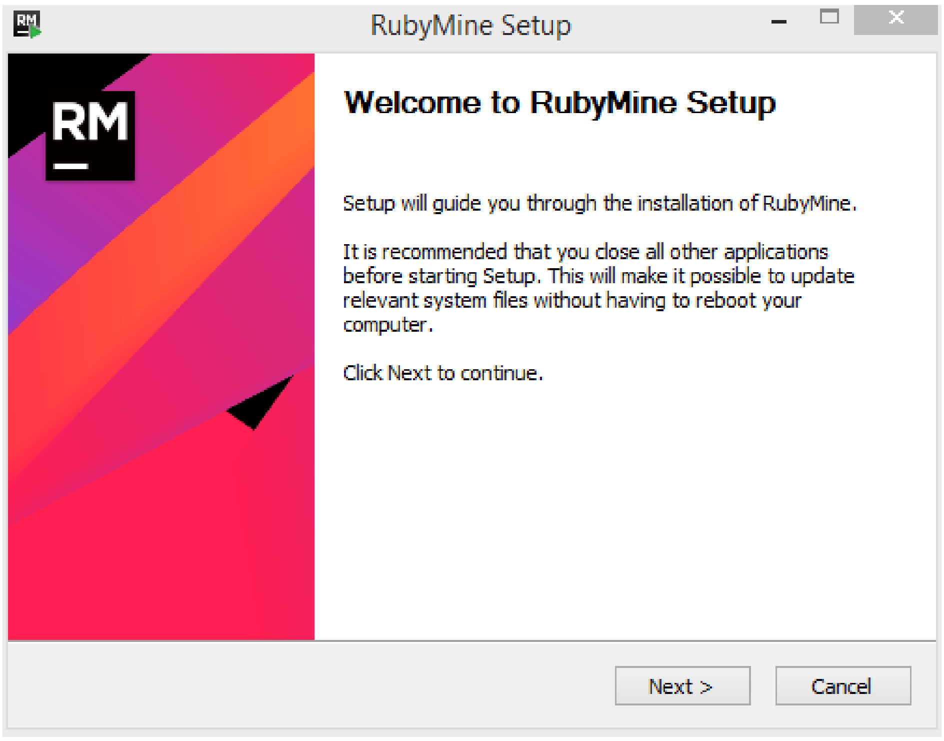
RubyMine IDE

We will use RubyMine IDE to write and execute our test cases. The official website for RubyMine IDE is https://www.jetbrains.com/ruby/.



RubyMine is a paid IDE with a 30-day free trial. Click on free trial button to download. You may also use some other IDE of your choice to write the test-cases.

On-click of the free trial button, it will start downloading. Once the download is done, install it on your system.



RubyMine Setup

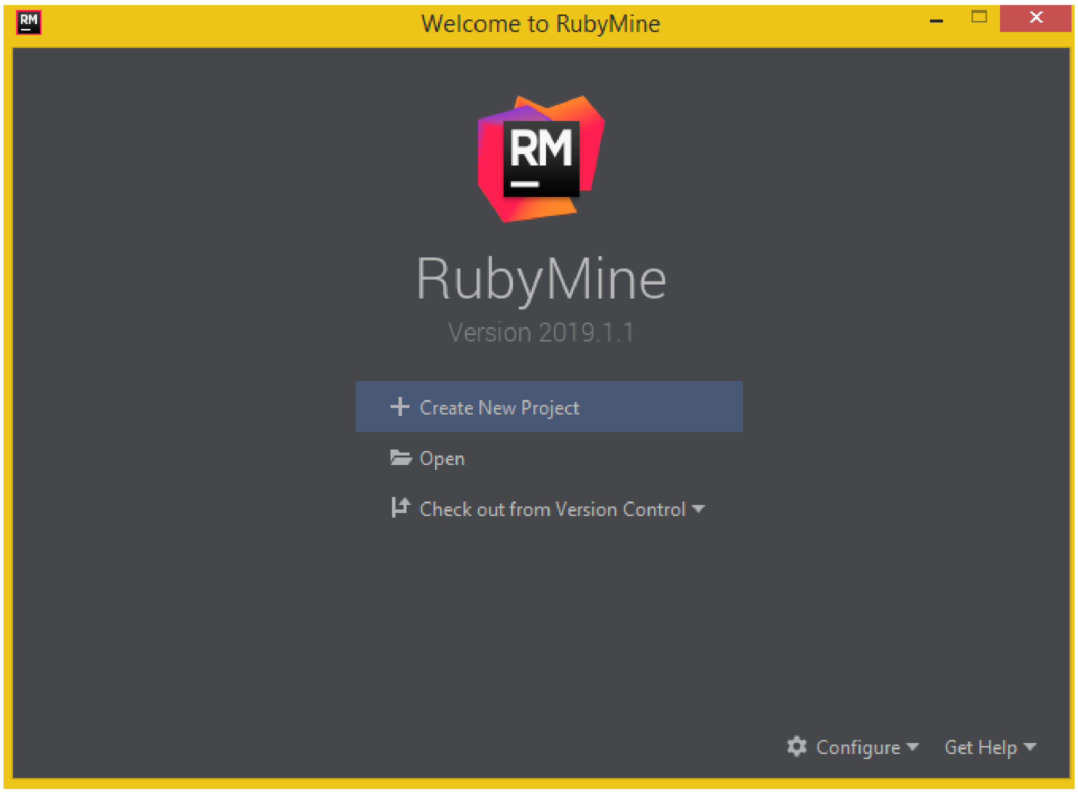
Welcome to RubyMine Setup

Setup will guide you through the installation of RubyMine.

It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

Click Next to continue.

Click on Next and proceed with the installation. Once the installation is complete, open the IDE and it will ask you to create the project as shown below:

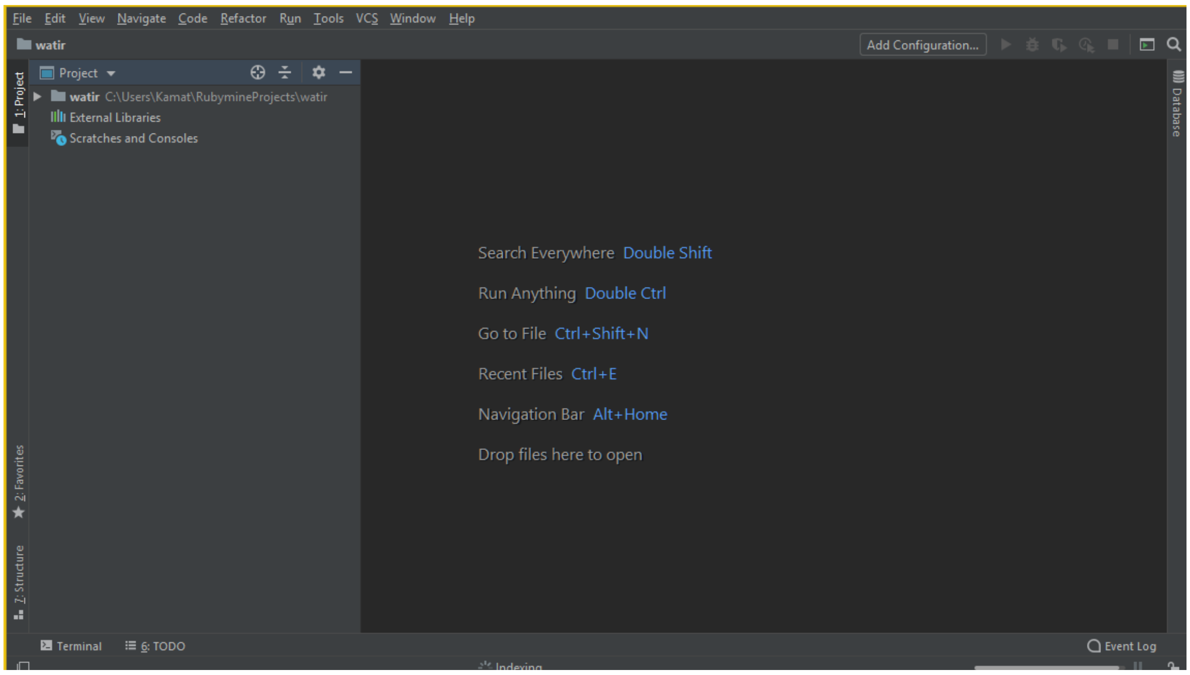


Welcome to RubyMine

RubyMine

Version 2019.1.1

Click on Create New Project, we have entered the name of the project as Watir and here is the display in the IDE:



Search Everywhere Double Shift

Run Anything Double Ctrl

Go to File Ctrl+Shift+N

Recent Files Ctrl+E

Navigation Bar Alt+Home

Drop files here to open

We will write our test-cases in watir/ folder and execute it.

4. Watir — Installing Drivers for Browsers

In this chapter, we are going to install browser drivers that we need to test our project using Watir. Prior to Watir 6, we had to include *watir-webdriver* to use the browser drivers. With the release of Watir 6, the *watir-webdriver* is available as part of Watir package and users do not have to add the *watir-webdriver* separately.

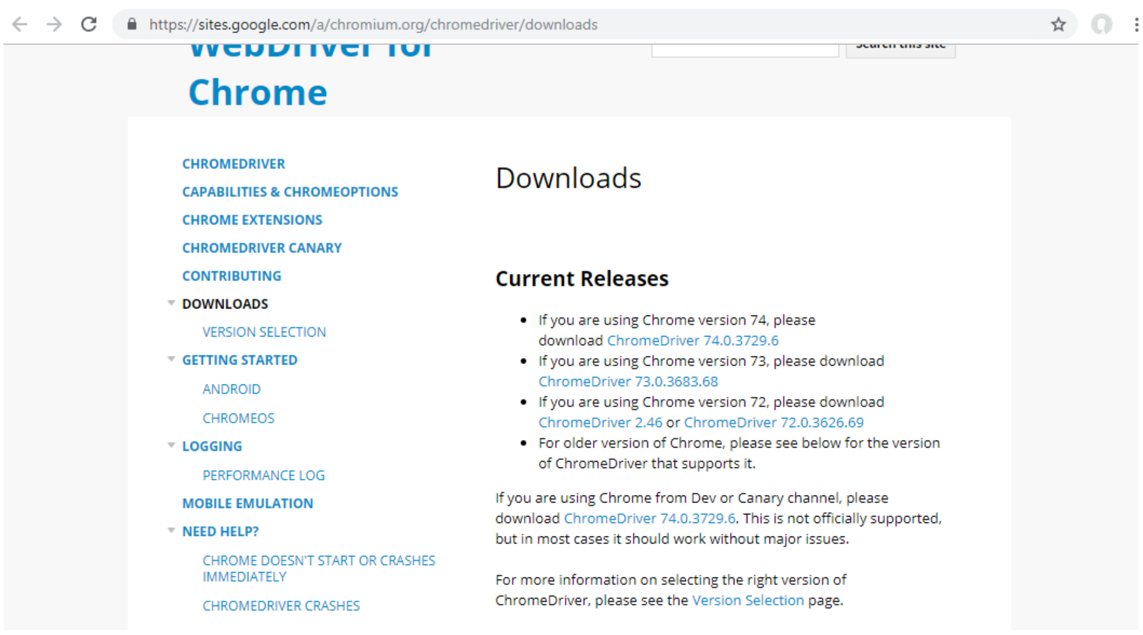
The browsers like Chrome, Firefox , and Safari are available by default and you do not have to add them externally. Incase while testing you get an issue that the driver for the browser is not available, you can always download them as instructed below.

We are going to install drivers for following browsers:

- Driver for Browsers - Chrome
- Driver for Browsers - Firefox
- Driver for Browsers - Edge
- Driver for Browsers - Internet Explorer
- Driver for Browsers - Safari

Driver for Browser - Chrome

To get the driver for chrome browser, go to: https://sites.google.com/a/chromium.org/chromedriver/downloads



Downloads

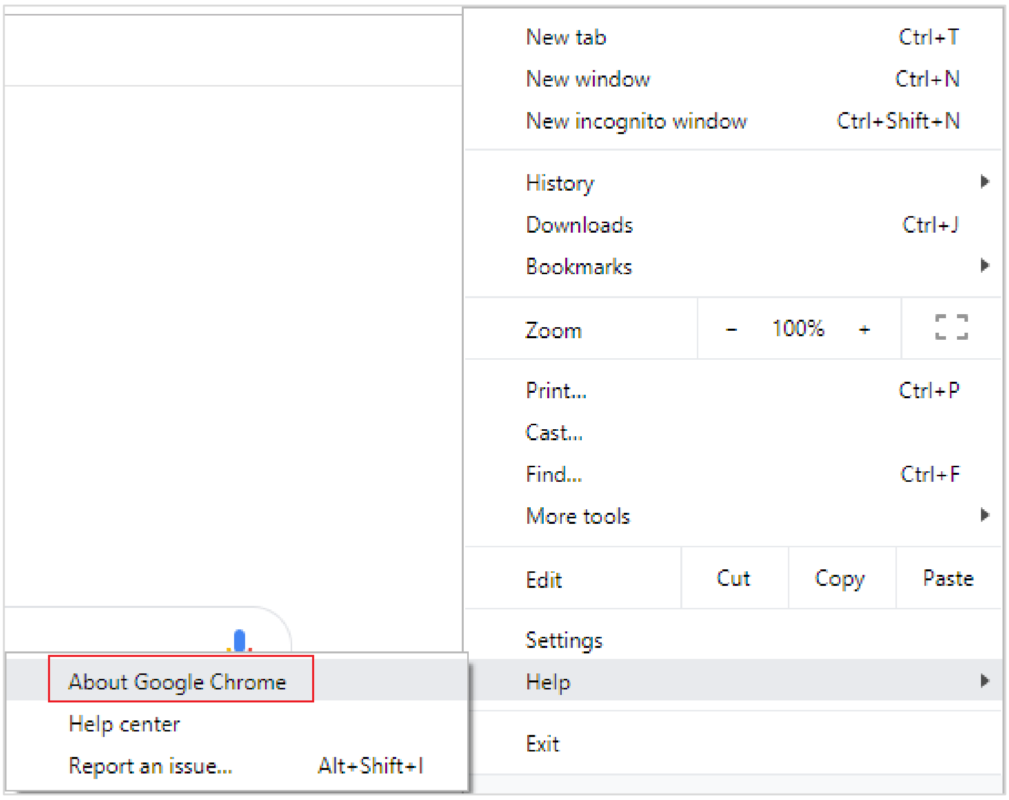
Current Releases

- If you are using Chrome version 74, please download ChromeDriver 74.0.3729.6
- If you are using Chrome version 73, please download ChromeDriver 73.0.3683.68
- If you are using Chrome version 72, please download ChromeDriver 2.46 or ChromeDriver 72.0.3626.69
- For older version of Chrome, please see below for the version of ChromeDriver that supports it.

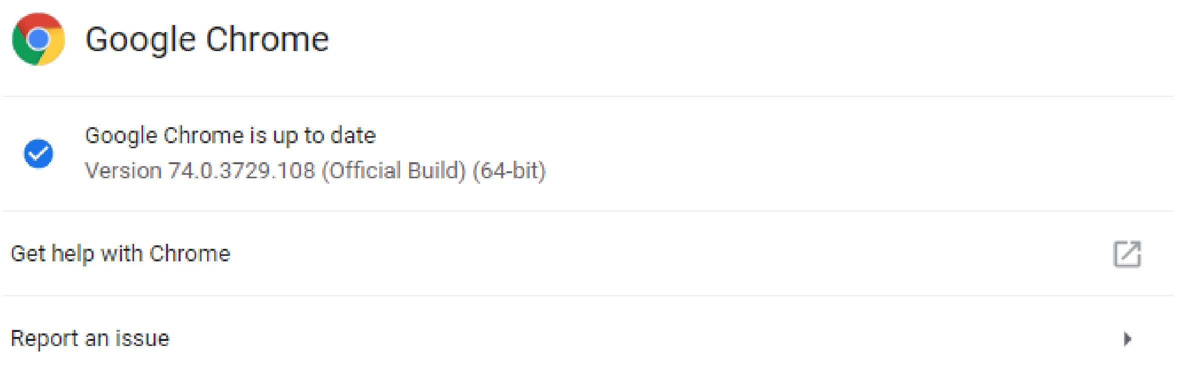
If you are using Chrome from Dev or Canary channel, please download ChromeDriver 74.0.3729.6. This is not officially supported, but in most cases it should work without major issues.

For more information on selecting the right version of ChromeDriver, please see the Version Selection page.

Check the version of your browser and accordingly download the Chrome driver. To check the version of your chrome browser, do as shown here:

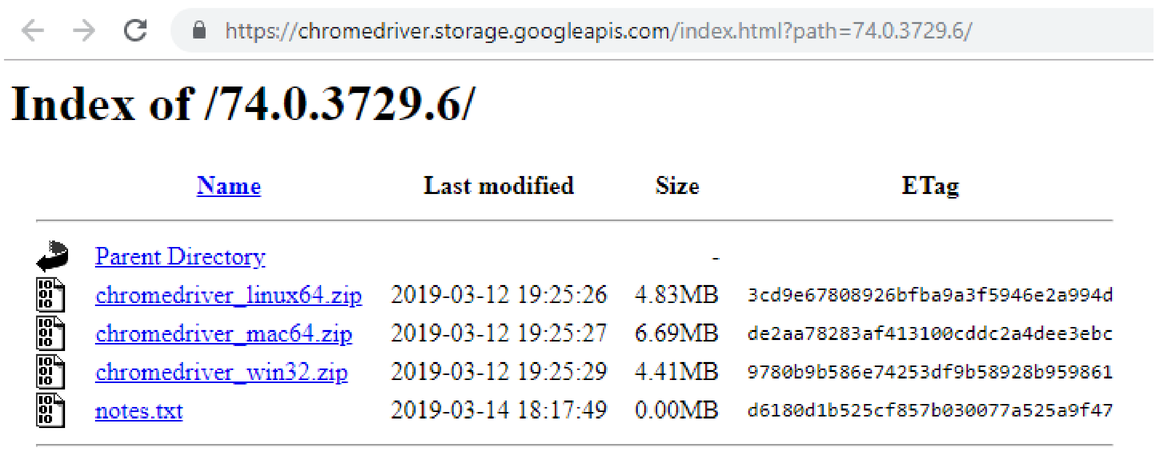


Click on About Google Chrome and it will give you the chrome version as shown below:



Google Chrome is up to date
Version 74.0.3729.108 (Official Build) (64-bit)

So our version is 74. So, download chrome driver version 74.



https://chromedriver.storage.googleapis.com/index.html?path=74.0.3729.6/

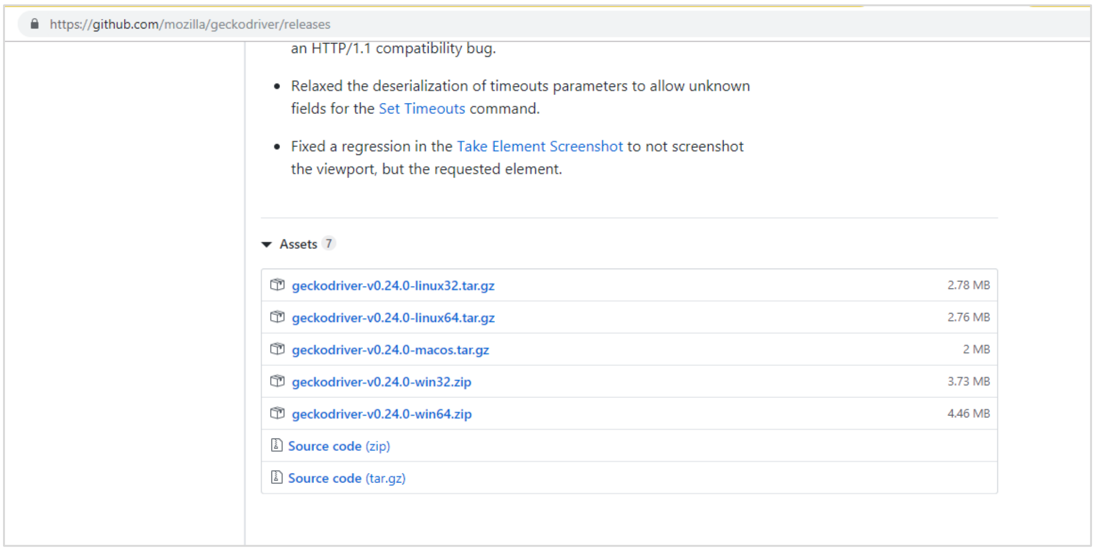
Index of /74.0.3729.6/

Name	Last modified	Size	ETag
Parent Directory		-	
chromedriver_linux64.zip	2019-03-12 19:25:26	4.83MB	3cd9e67808926bfba9a3f5946e2a994d
chromedriver_mac64.zip	2019-03-12 19:25:27	6.69MB	de2aa78283af413100cddc2a4dee3ebc
chromedriver_win32.zip	2019-03-12 19:25:29	4.41MB	9780b9b586e74253df9b58928b959861
notes.txt	2019-03-14 18:17:49	0.00MB	d6180d1b525cf857b030077a525a9f47

Next, download the chrome driver depending on your operating system. We will download chromedriver_win32.zip, it is meant for both 32-bit and 64-bit machines. In case you are planning to use the driver downloaded, add the location somewhere on your PATH variable.

Driver for Browser - Firefox

For Firefox driver, go to: https://github.com/mozilla/geckodriver/releases as shown in the screenshot below:



https://github.com/mozilla/geckodriver/releases

an HTTP/1.1 compatibility bug.

- Relaxed the deserialization of timeouts parameters to allow unknown fields for the Set Timeouts command.
- Fixed a regression in the Take Element Screenshot to not screenshot the viewport, but the requested element.

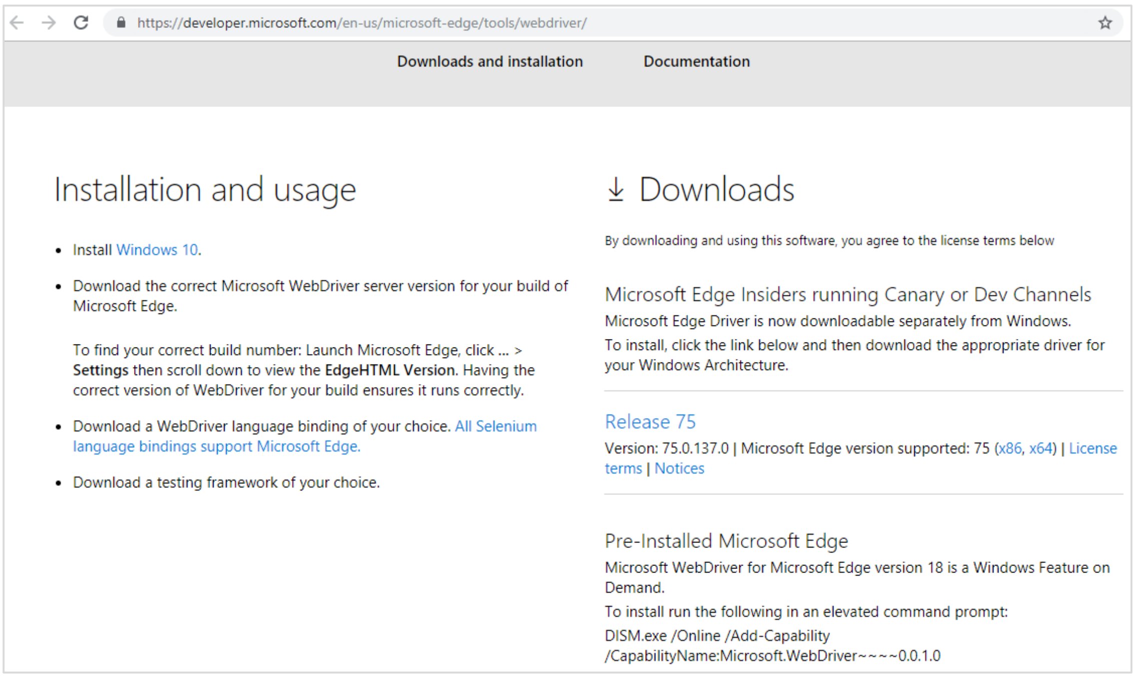
Assets 7

geckodriver-v0.24.0-linux32.tar.gz	2.78 MB
geckodriver-v0.24.0-linux64.tar.gz	2.76 MB
geckodriver-v0.24.0-macos.tar.gz	2 MB
geckodriver-v0.24.0-win32.zip	3.73 MB
geckodriver-v0.24.0-win64.zip	4.46 MB
Source code (zip)	
Source code (tar.gz)	

As per your operating system, download the driver as shown above. In case you planning to use the driver downloaded, add the location somewhere on your PATH variable.

Driver for Browser - Edge

To use the driver go to: https://developer.microsoft.com/en-us/microsoft-edge/tools/webdriver/ as shown below:


https://developer.microsoft.com/en-us/microsoft-edge/tools/webdriver/
Downloads and installation
Documentation

Installation and usage

- Install Windows 10.
- Download the correct Microsoft WebDriver server version for your build of Microsoft Edge.

  To find your correct build number: Launch Microsoft Edge, click ... > **Settings** then scroll down to view the **EdgeHTML Version**. Having the correct version of WebDriver for your build ensures it runs correctly.
- Download a WebDriver language binding of your choice. All Selenium language bindings support Microsoft Edge.
- Download a testing framework of your choice.

Downloads

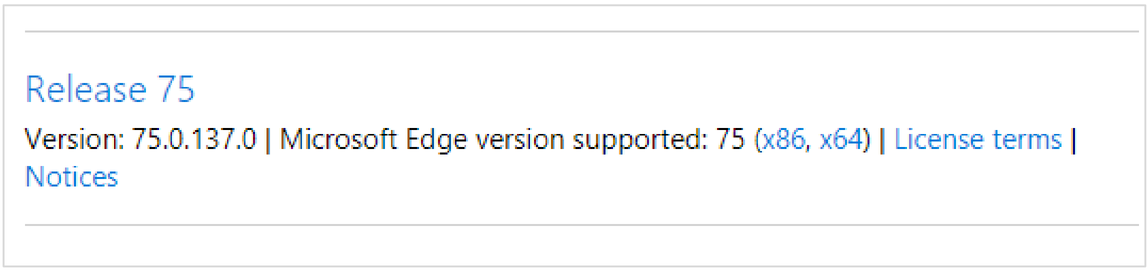
By downloading and using this software, you agree to the license terms below

Microsoft Edge Insiders running Canary or Dev Channels
Microsoft Edge Driver is now downloadable separately from Windows.
To install, click the link below and then download the appropriate driver for your Windows Architecture.

Release 75
Version: 75.0.137.0 | Microsoft Edge version supported: 75 (x86, x64) | License terms | Notices

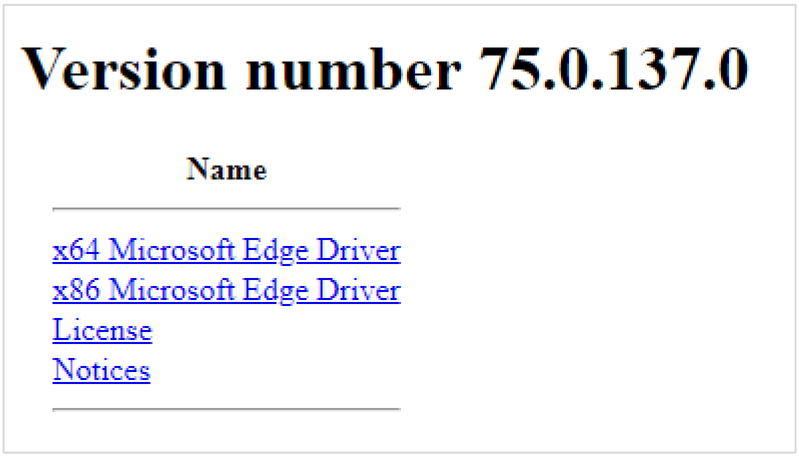
Pre-Installed Microsoft Edge
Microsoft WebDriver for Microsoft Edge version 18 is a Windows Feature on Demand.
To install run the following in an elevated command prompt:
DISM.exe /Online /Add-Capability /CapabilityName:Microsoft.WebDriver~~~~0.0.1.0

Click on the Version of Microsoft Edge you have as shown below:



Release 75
Version: 75.0.137.0 | Microsoft Edge version supported: 75 (x86, x64) | License terms | Notices

Click on Release link and it will redirect you to:



Version number 75.0.137.0

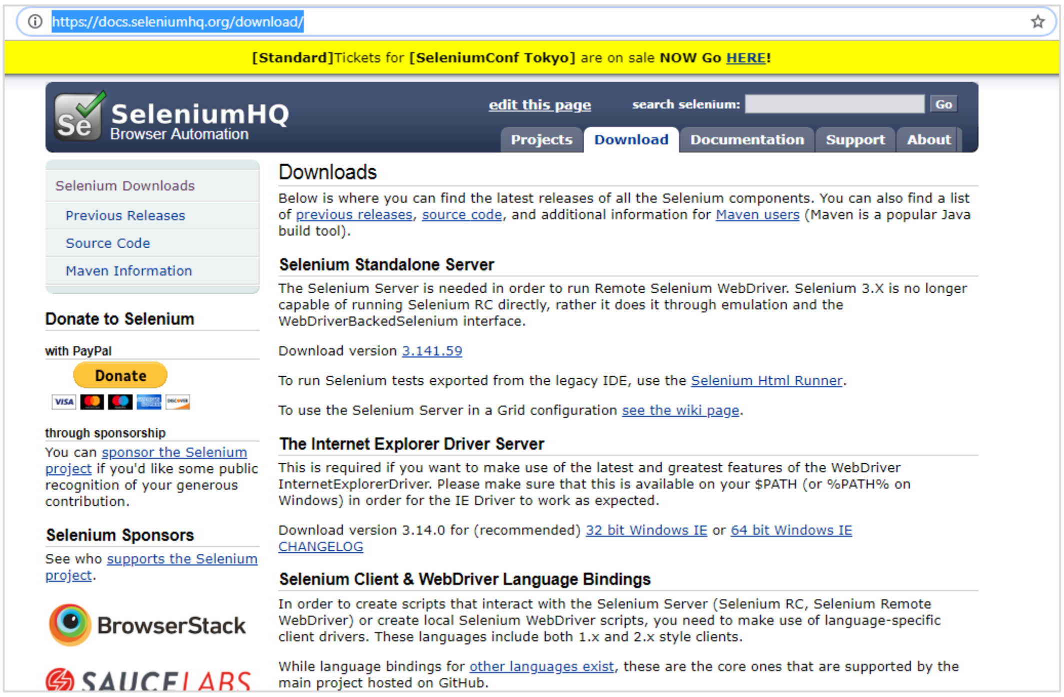
Name

x64 Microsoft Edge Driver
x86 Microsoft Edge Driver
License
Notices

Here, you can find the download version available for windows. In case you are planning to use the driver downloaded, add the location somewhere on your PATH variable.

Driver for Browser - Internet Explorer

To install the driver for Internet Explorer go to: https://docs.seleniumhq.org/download/ as shown below:



https://docs.seleniumhq.org/download/

[Standard]Tickets for [SeleniumConf Tokyo] are on sale NOW Go HERE!

Donate to Selenium

with PayPal

through sponsorship

You can sponsor the Selenium project if you'd like some public recognition of your generous contribution.

Selenium Sponsors

See who supports the Selenium project.

Downloads

Below is where you can find the latest releases of all the Selenium components. You can also find a list of previous releases, source code, and additional information for Maven users (Maven is a popular Java build tool).

Selenium Standalone Server

The Selenium Server is needed in order to run Remote Selenium WebDriver. Selenium 3.X is no longer capable of running Selenium RC directly, rather it does it through emulation and the WebDriverBackedSelenium interface.

Download version 3.141.59

To run Selenium tests exported from the legacy IDE, use the Selenium Html Runner.

To use the Selenium Server in a Grid configuration see the wiki page.

The Internet Explorer Driver Server

This is required if you want to make use of the latest and greatest features of the WebDriver InternetExplorerDriver. Please make sure that this is available on your $PATH (or %PATH% on Windows) in order for the IE Driver to work as expected.

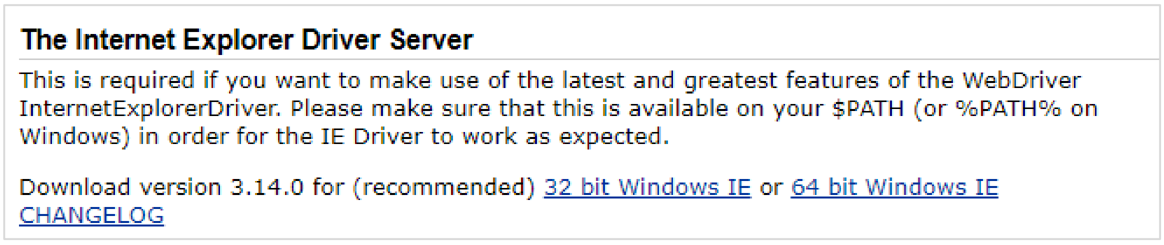
Download version 3.14.0 for (recommended) 32 bit Windows IE or 64 bit Windows IE
CHANGELOG

Selenium Client & WebDriver Language Bindings

In order to create scripts that interact with the Selenium Server (Selenium RC, Selenium Remote WebDriver) or create local Selenium WebDriver scripts, you need to make use of language-specific client drivers. These languages include both 1.x and 2.x style clients.

While language bindings for other languages exist, these are the core ones that are supported by the main project hosted on GitHub.

The details for the download section on IE are shown here:



The Internet Explorer Driver Server

This is required if you want to make use of the latest and greatest features of the WebDriver InternetExplorerDriver. Please make sure that this is available on your $PATH (or %PATH% on Windows) in order for the IE Driver to work as expected.

Download version 3.14.0 for (recommended) 32 bit Windows IE or 64 bit Windows IE
CHANGELOG

As per your OS, download the 32 or 64 bit windows IE. Also add the location to you PATH variable to make use of the driver.

Drivers for Browsers - Safari

The drivers for safari are not required to be downloaded externally for safari version 10 onwards. They are downloaded along with the gem package while installing Watir.

Please note that drivers for Chrome, Firefox , Internet Explorer are also available when Watir is installed. But incase if you face any issues and need a specific version to be tested, you can download them as per your browser and update the location in PATH to make use of it in testing your application.

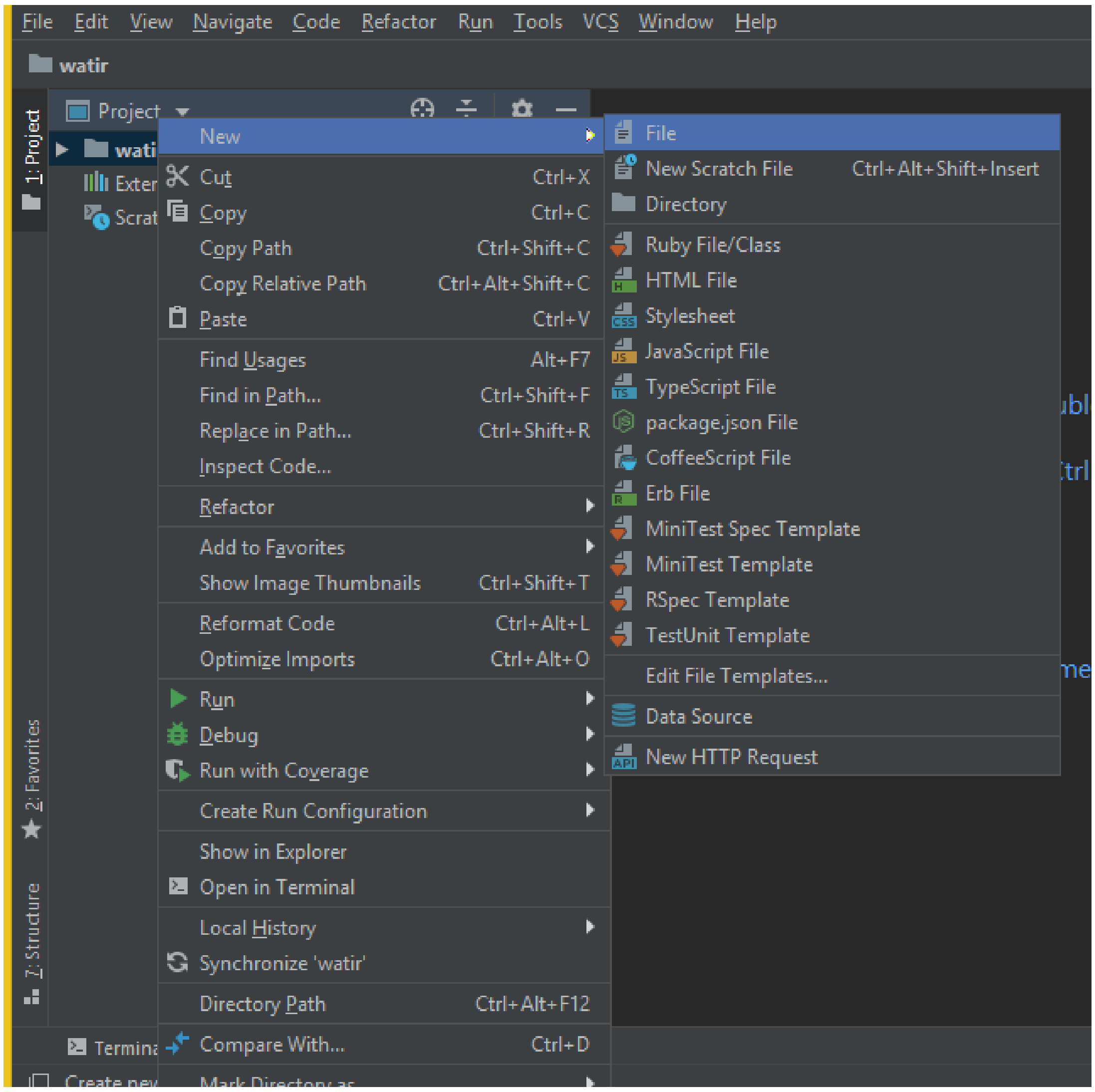
5. Watir — Working with Browsers

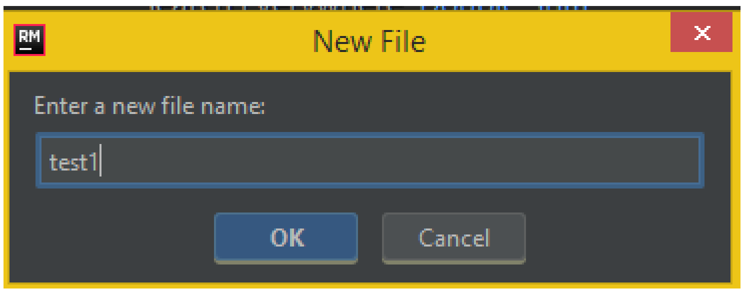
By default, Watir will open chrome browser in-case the browser name is not specified. The required browser drivers are installed along with Watir installation. In case you face any issues working with browsers, install the driver as shown in the Browsers drivers chapter and update the location in PATH variable.

In this chapter, we will understand how to open the browser using Watir.

Steps to Open a Browser using Watir

Open the IDE RubyMine and create a new file: test1.rb

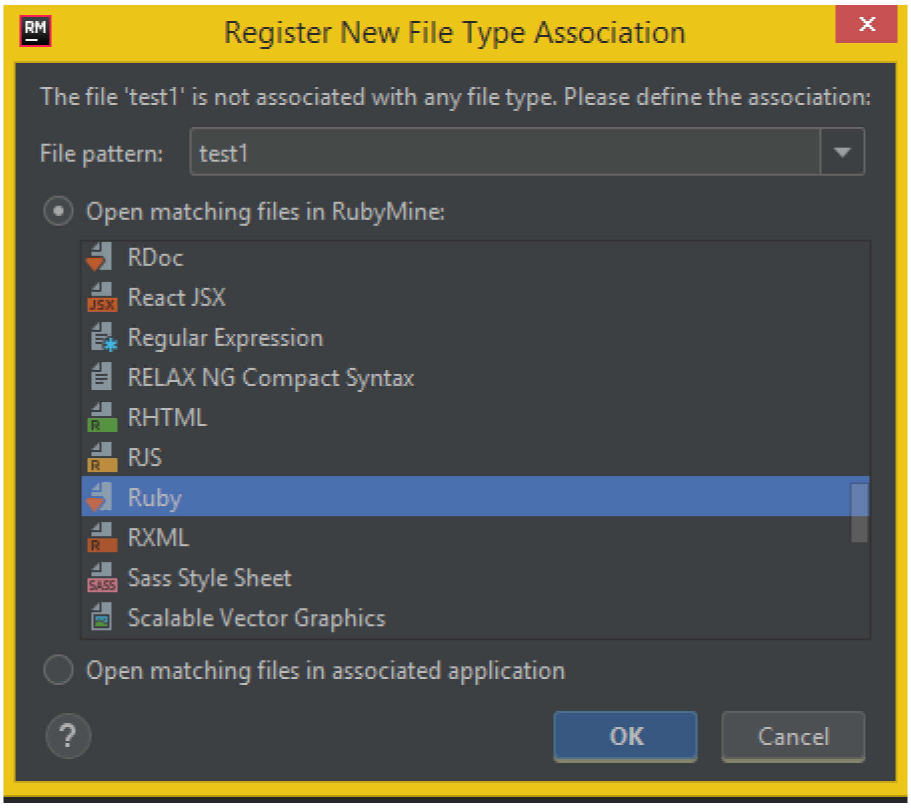




Enter a new file name:

test1

Select OK and click the file pattern as ruby as shown below:



The file 'test1' is not associated with any file type. Please define the association:

File pattern: test1

(●) Open matching files in RubyMine: Ruby

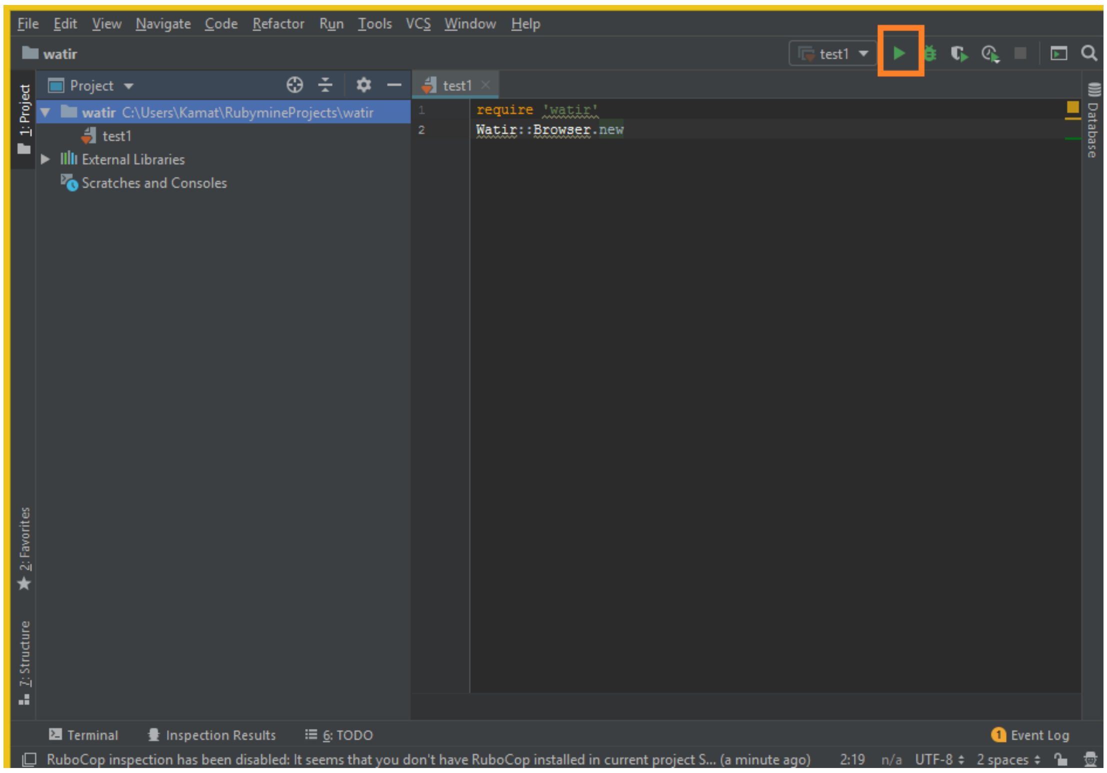
( ) Open matching files in associated application

Click on OK to create the file.

Now we will write a simple code that will open the browser as shown below:

test1.rb

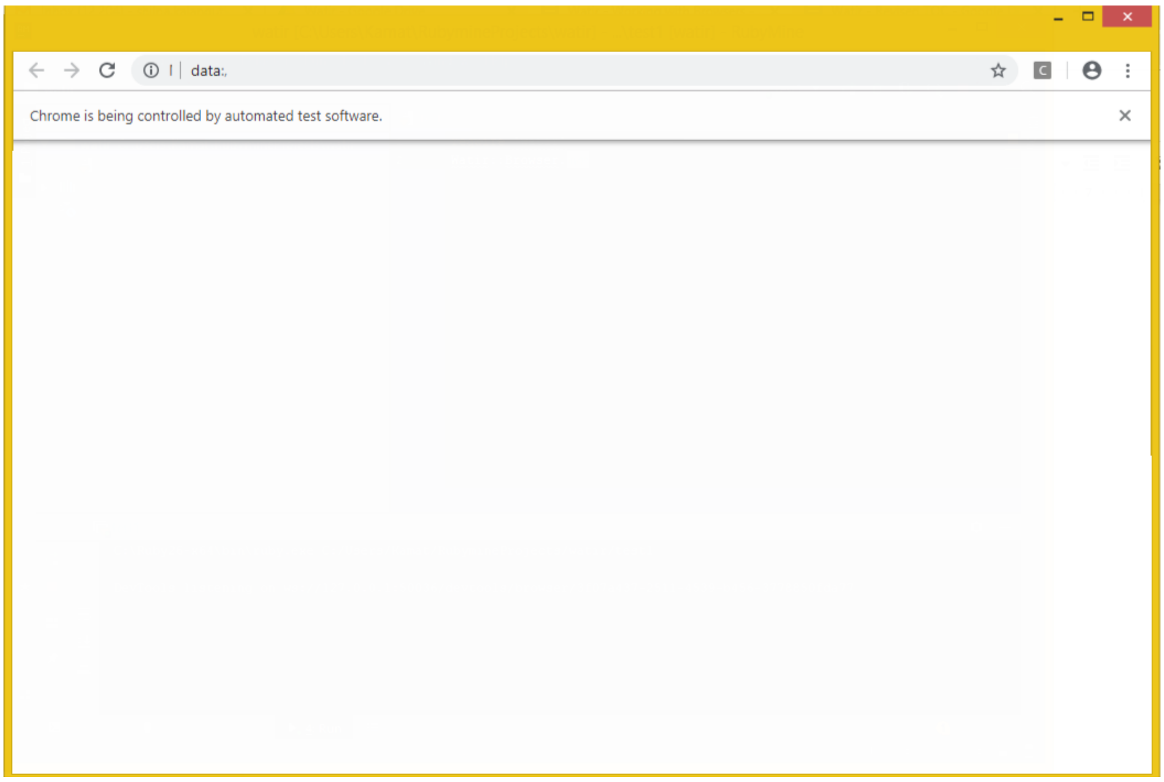
```
require 'watir'
Watir::Browser.new
```


File Edit View Navigate Code Refactor Run Tools VCS Window Help
watir
test1
Project
watir C:\Users\Kamat\RubymineProjects\watir
test1
External Libraries
Scratches and Consoles
Terminal
Inspection Results
6: TODO
Event Log

```
require 'watir'
Watir::Browser.new
```

RuboCop inspection has been disabled: It seems that you don't have RuboCop installed in current project S... (a minute ago)   2:19   n/a   UTF-8   2 spaces

Click on the Run button that is highlighted in the IDE as shown above. On-click of Run, it will open browser as shown below:



Chrome is being controlled by automated test software.

The browser will open and close automatically. Let us now add some more code to the test1.rb.

We can specify the name of the browser as shown below:

Example for Chrome

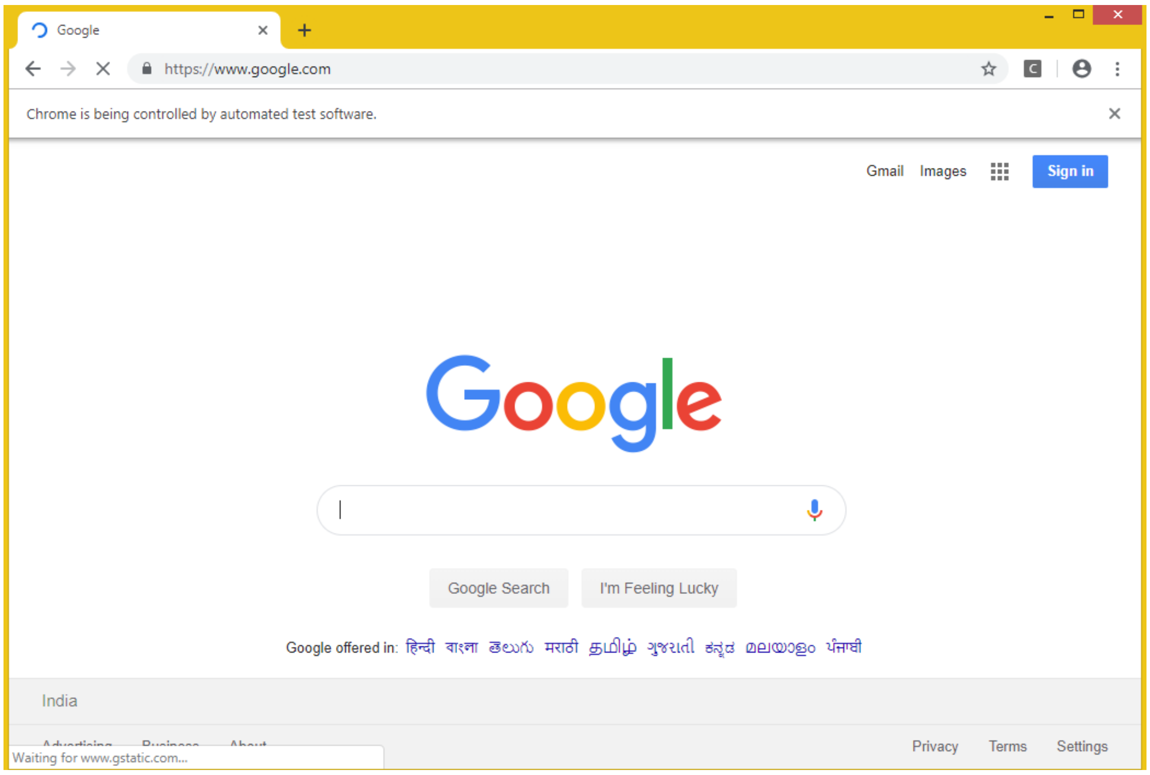
```
require 'watir'
Watir::Browser.new :chrome
```

Now let us open a page-url in our test case.

Example

```
require 'watir'
browser = Watir::Browser.new
browser.goto("https://www.google.com")
```

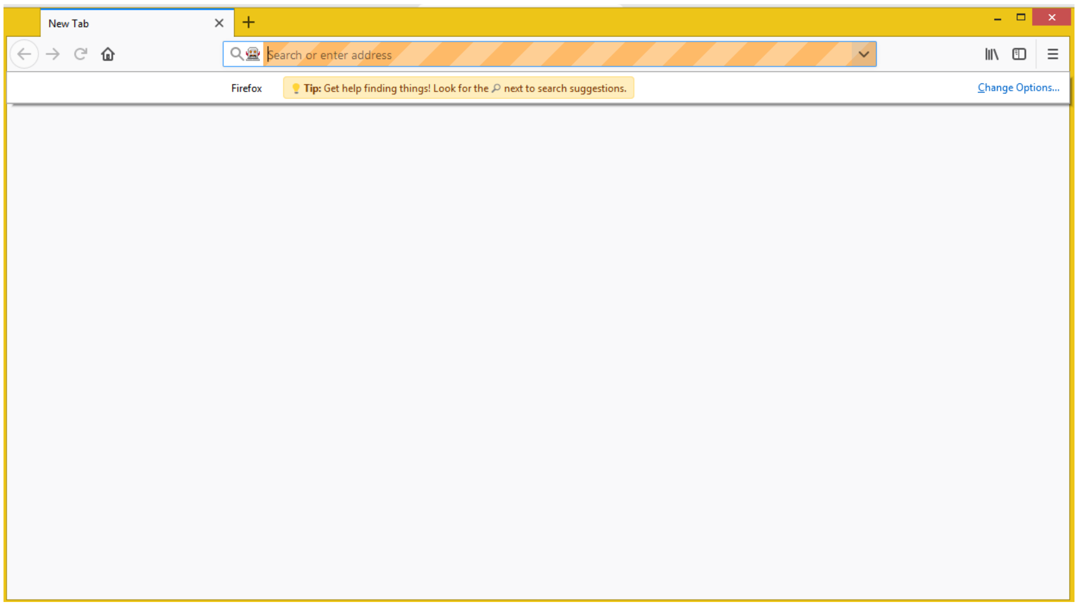
Click on Run to see the output as shown below:


Google
https://www.google.com
Chrome is being controlled by automated test software.
Gmail
Images
Sign in
Google
Google Search
I'm Feeling Lucky
Google offered in: हिन्दी বাংলা తెలుగు मराठी தமிழ் ગુજરાતી ಕನ್ನಡ മലയാളം ਪੰਜਾਬੀ
India
Privacy
Terms
Settings
Waiting for www.gstatic.com...

Similarly, you can open firefox, safari, Internet explorer browser.

Example for Firefox

```
require 'watir'
Watir::Browser.new :firefox
```


New Tab
Search or enter address
Firefox
Tip: Get help finding things! Look for the next to search suggestions.
Change Options...

Example for Internet Explorer

Watir Code

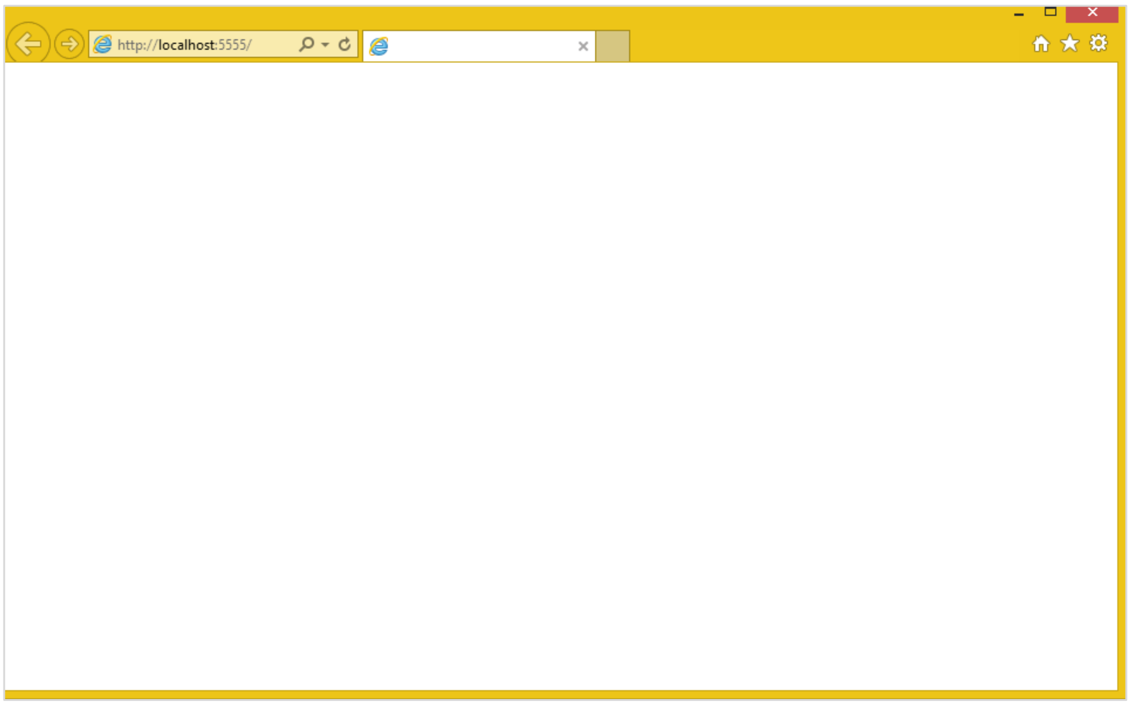
```
require 'watir'

browser = Watir::Browser.new :ie

browser.goto("https://www.google.com")
```

When we run the code following error is displayed:

```
Unable to find IEDriverServer. Please download the server from
(Selenium::WebDriver::Error::WebDriverError)

http://selenium-release.storage.googleapis.com/index.html and place it
somewhere on your PATH.

More info at
https://github.com/SeleniumHQ/selenium/wiki/InternetExplorerDriver.
```

This means that watir package does not have InternetExplorer Driver. We have downloaded the same from here : https://docs.seleniumhq.org/download/ and updated in PATH variable.

Now run it again to see the Internet Explorer browser opening as shown below:



Watir code to open Safari Browser

```
require 'watir'
browser = Watir::Browser.new :safari
browser.goto("https://www.google.com")
```

Watir code to Microsoft Edge browser

```
require 'watir'
browser = Watir::Browser.new :edge
browser.goto("https://www.google.com")
```

6. Watir — Web Elements

In this chapter, we will discuss how to work with following in Watir:

- Working with Textboxes
- Working with Combos
- Working with Radio Buttons
- Working with Checkboxes
- Working with Buttons
- Working with Links
- Working with Div’s

Working with Textboxes

Syntax

```
browser.text_field id: 'firstname' // will get the reference of the textbox
```

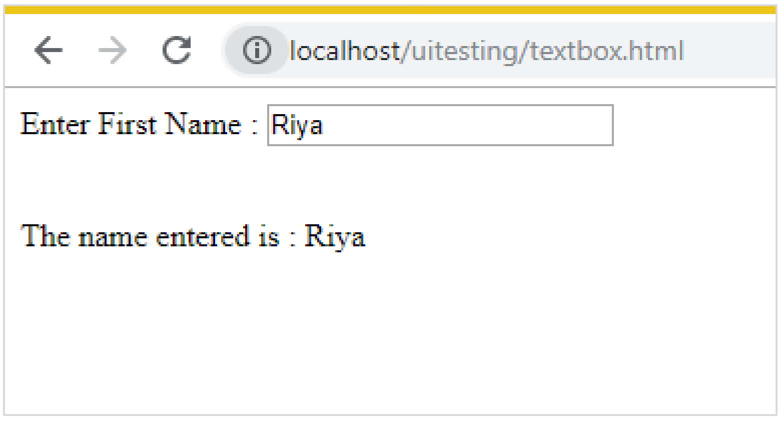
Here will try to understand how to work with textboxes on the UI.

Consider the page Textbox.html as shown below:

```
<html>
<head>
<title>Testing UI using Watir</title>
</head>
<body>
<script type="text/javascript">
function wsentered() {
console.log("inside wsentered");
     var firstname = document.getElementById("firstname");
     if (firstname.value != "") {
            document.getElementById("displayfirstname").innerHTML = "The name
entered is : " + firstname.value;
            document.getElementById("displayfirstname").style.display = "";
     }
}
</script>
<div id="divfirstname">
```

```
     Enter First Name : <input type="text" id="firstname" name="firstname" onchange="wsentered()" />
</div>
<br/>
<br/>
<div style="display:none;" id="displayfirstname">
</div>
</body>
</html>
```

The corresponding output is as shown below:


localhost/uitesting/textbox.html

Enter First Name : Riya

The name entered is : Riya

We are having a textbox, when you enter the name onchange event is fired and the name is displayed below.

Now let us write the code, wherein we will locate the textbox and enter the name and fire the onchange event.

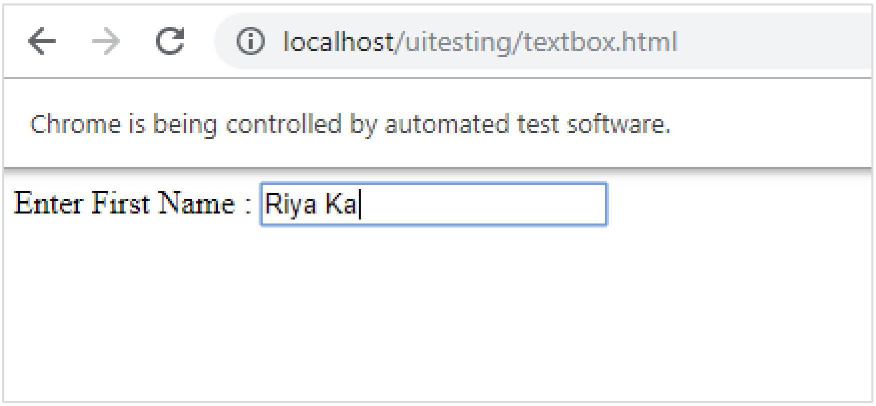
Watir Code

```
require 'watir'
b = Watir::Browser.new :chrome
b.goto('http://localhost/uitesting/textbox.html')
t = b.text_field id: 'firstname'
t.exists?
t.set 'Riya Kapoor'
t.value
t.fire_event('onchange')
```

We are using chrome browser and given the pageurl as http://localhost/uitesting/textbox.html.

Using *goto* api browser will open the pageurl and we are going to find text_field having id: firstname. If that exists, we will set value as Riya Kapoor and will use *fire_event* api to fire the onchange event.

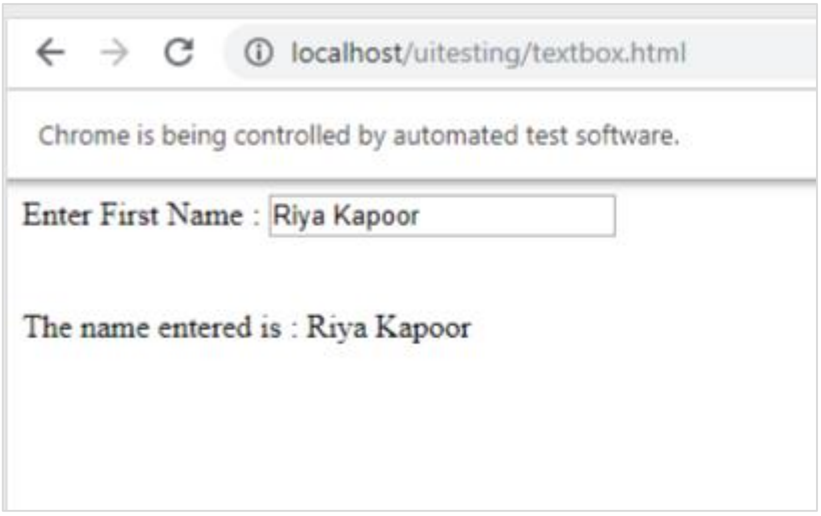
Now, let us run the code to display the output as shown below:



localhost/uitesting/textbox.html

Chrome is being controlled by automated test software.

Enter First Name : Riya Ka



localhost/uitesting/textbox.html

Chrome is being controlled by automated test software.

Enter First Name : Riya Kapoor

The name entered is : Riya Kapoor

Working with Combos

Syntax

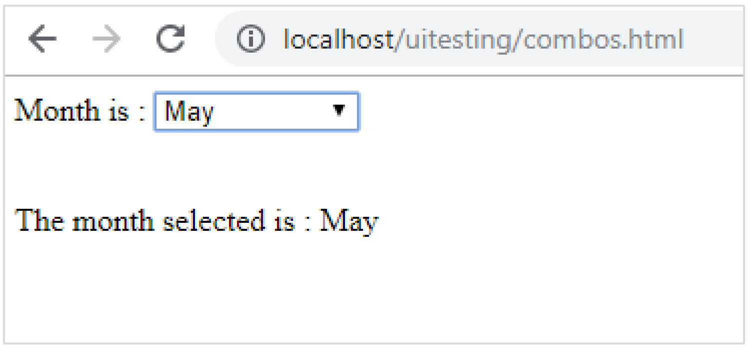
```
browser.select_list id: 'months' // will get the reference of the dropdown
```

The test page that we are going to test now is shown here:

```
<html>
<head>
<title>Dropdown</title>
</head>
<body>
<script type="text/javascript">
```

```
function wsselected() {
      var months = document.getElementById("months");
      if (months.value != "") {
             document.getElementById("displayselectedmonth").innerHTML = "The
month selected is : " + months.value;
             document.getElementById("displayselectedmonth").style.display = "";
      }
}
</script>
<form name="myform" method="POST">
<div>
Month is :
<select name="months" id="months" onchange="wsselected()">
<option value="">Select Month</option>
<option value="Jan">January</option>
<option value="Feb">February</option>
<option value="Mar">March</option>
<option value="Apr">April</option>
<option value="May">May</option>
<option value="Jun">June</option>
<option value="Jul">July</option>
<option value="Aug">August</option>
<option value="Sept">September</option>
<option value="Oct">October</option>
<option value="Nov">November</option>
<option value="Dec">December</option>
</select>
</div>
<br/>
<br/>
<div style="display:none;" id="displayselectedmonth">
</div>
</body>
</html>
```

Output



localhost/uitesting/combos.html

Month is : May

The month selected is : May

When you select month from the dropdown the same is displayed below.

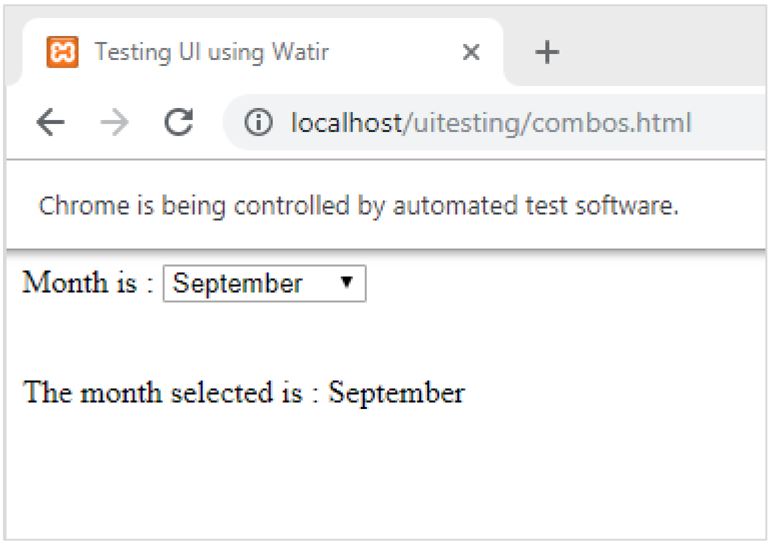
Let us now test the same using Watir.

Watir Code for combo selection

```
require 'watir'
b = Watir::Browser.new :chrome
b.goto('http://localhost/uitesting/combos.html')
t = b.select_list id: 'months'
t.exists?
t.select 'September'
t.selected_options
t.fire_event('onchange')
```

To work with combos, you need to locate the select element using *b.select_list* api followed by the id of the dropdown. To select the value from the dropdown, you need to use *t.select* and the value you want.

The output on execution is as follows:



localhost/uitesting/combos.html

Chrome is being controlled by automated test software.

Month is : September

The month selected is : September

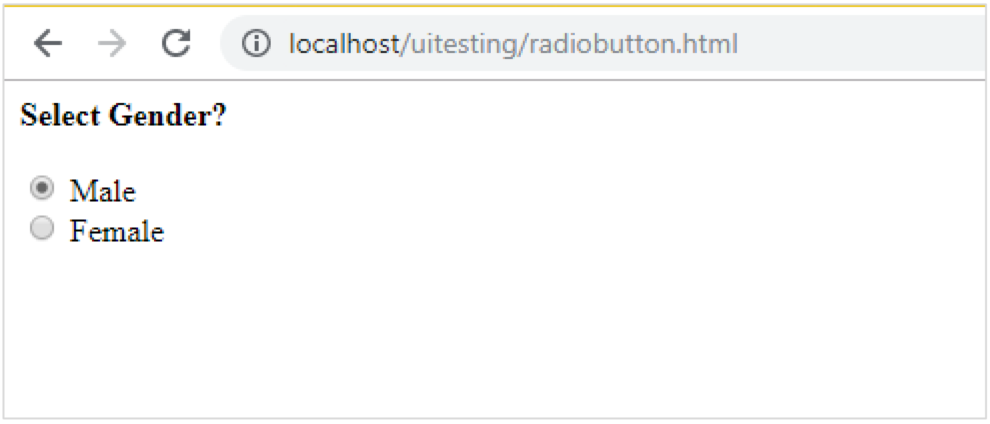
Working with Radio Buttons

Syntax

```
browser.radio value: 'female'  // will get the reference of the radio button
with value “female”
```

Here is a test page that we will use to work with radio buttons:

```
<html>
<head>
<title>Testing UI using Watir</title>
</head>
<body>
<form name="myform" method="POST">
<b>Select Gender?</b>
<div><br/>
<input type="radio" name="gender" value="male" checked> Male<br/>
<input type="radio" name="gender" value="female"> Female<br/>
</div>
</form>
</body>
</html>
```



Select Gender?

● Male
○ Female

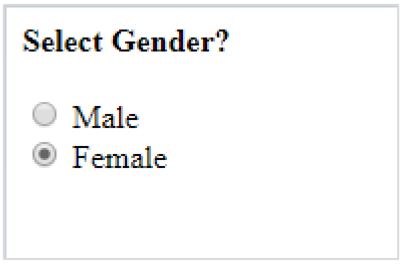
We will select radio button with value Female as shown in the Watir code:

```
require 'watir'
b = Watir::Browser.new
b.goto('http://localhost/uitesting/radiobutton.html')
t = b.radio value: 'female'
t.exists?
```

```
t.set
b.screenshot.save 'radiobutton.png'
```

To work with radio button, we need to tell the browser about the value we are selecting i.e. **b.radio value:"female"**

We are also taking the screenshot and saved that as radiobutton.png and the same is displayed below:



Working with Checkboxes

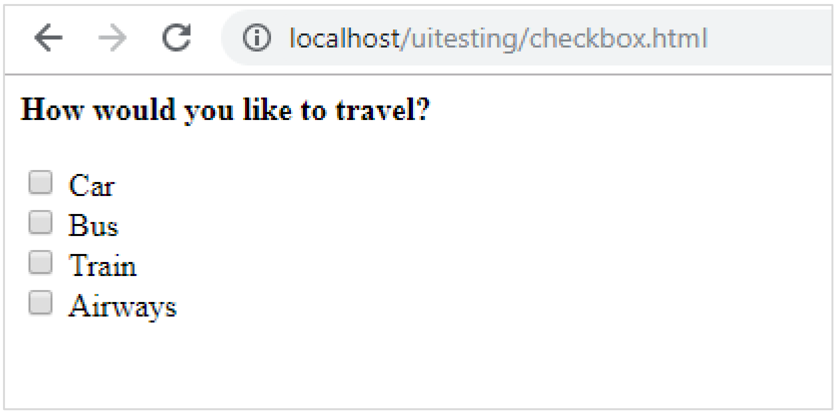
Syntax

```
browser. checkbox value: 'Train'  // will get the reference of the checkbox
with value "Train"
```

Here is the test page for checkboxes:

```
<html>
<head>
<title>Testing UI using Watir</title>
</head>
<body>
<form name="myform" method="POST">
<b>How would you like to travel?</b>
<div><br>
<input type="checkbox" name="option1" value="Car"> Car<br>
<input type="checkbox" name="option2" value="Bus"> Bus<br>
<input type="checkbox" name="option3" value="Train"> Train<br>
<input type="checkbox" name="option4" value="Air"> Airways<br>
<br>
</div>


</form>
</body>
```

```
</html>
```



localhost/uitesting/checkbox.html

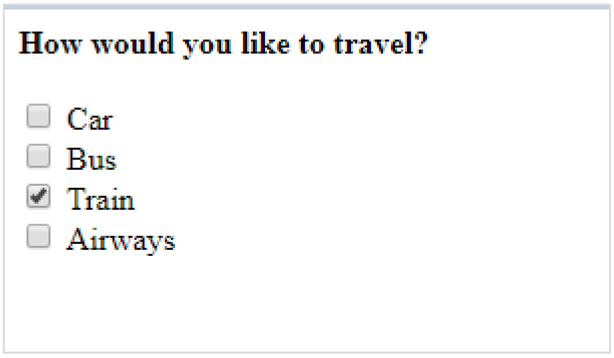
How would you like to travel?

- Car
- Bus
- Train
- Airways

Now, let us use Watir to locate the checkbox in browser as shown below:

```
require 'watir'
b = Watir::Browser.new
b.goto('http://localhost/uitesting/checkbox.html')
t = b.checkbox value: 'Train'
t.exists?
t.set
b.screenshot.save 'checkbox.png'
```

To locate checkbox in the browser, use *b.checkbox* with value you want to select.



How would you like to travel?

- Car
- Bus
- Train
- Airways

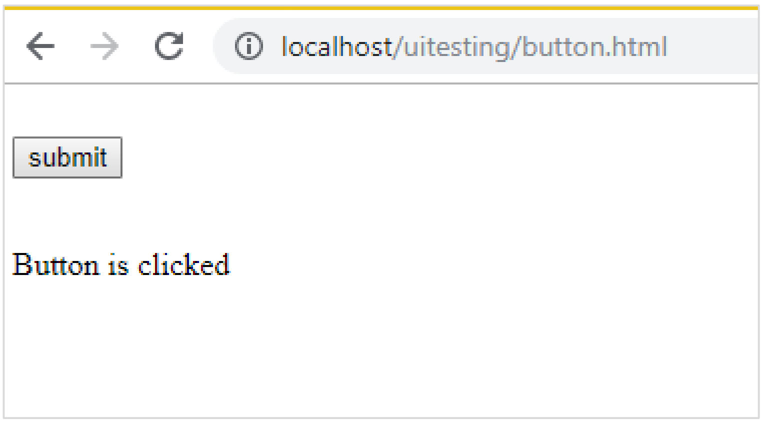
Working with Buttons

Syntax

```
browser.button(:name => "btnsubmit").click
 // will get the reference to the button element with has name “btnsubmit”
```

Here is the test page for button:

```
<html>
<head>
<title>Testing UI using Watir</title>
</head>
<body>
<script type="text/javascript">
function wsclick() {
     document.getElementById("buttondisplay").innerHTML = "Button is clicked";
     document.getElementById("buttondisplay").style.display = "";
}
</script>
<form name="myform" method="POST">
<div><br>
<input type="button" id="btnsubmit" name="btnsubmit"
value="submit"  onclick="wsclick()"/>
<br>
</div>
</form>
<br/>
<div style="display:none;" id="buttondisplay">
</div>
</body>
</html>
```

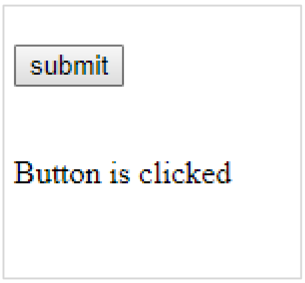


Button is clicked

Here is watir code to locate the button on the given page:

```
require 'watir'
b = Watir::Browser.new
b.goto('http://localhost/uitesting/button.html')
b.button(:name => "btnsubmit").click
b.screenshot.save 'button.png'
```

Here is the screenshot button.png



Button is clicked

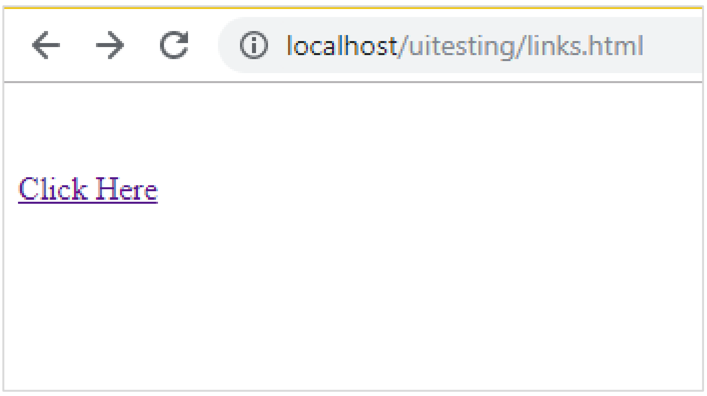
Working with Links

Syntax

```
browser.link text: 'Click Here'  // will get the reference  to the a tag with
text ‘Click Here’
```

We are going to use the following test page to test links:

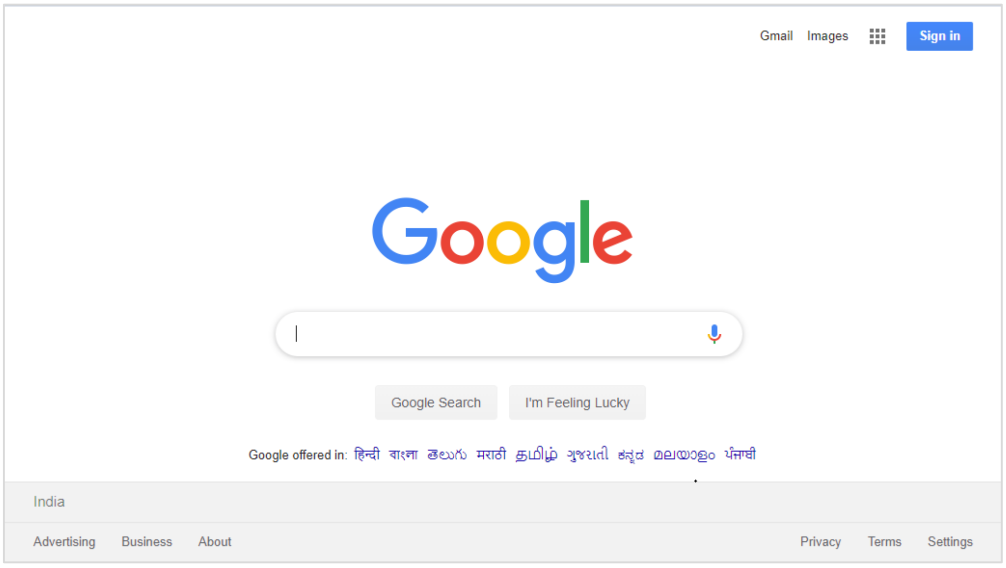
```
<html>
<head>
<title>Testing UI using Watir</title>
</head>
<body>
<br/>
<br/>
<a href="https://www.google.com">Click Here</a>
<br/>
</body>
</html>
```



The Watir details required to test links are as given below:

```
require 'watir'
b = Watir::Browser.new
b.goto('http://localhost/uitesting/links.html')
l = b.link text: 'Click Here'
l.click
b.screenshot.save 'links.png'
```

Output



Working with Div's

Syntax

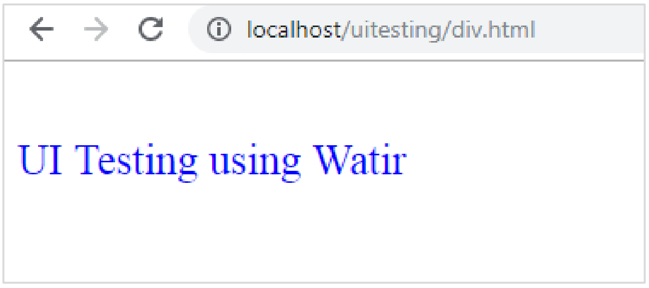
```
browser.div class: 'divtag'  // will get the reference  to div with class
“divtag”
```

Test page where we can test for div.

```
<html>
<head>
<title>Testing UI using Watir</title>
<style>
.divtag {
color: blue;
font-size: 25px;
}
</style>
</head>
<body>
<br/>
<br/>
<div class="divtag">
UI Testing using Watir
</div>
<br/>
```

```
</body>
</html>
```

Output

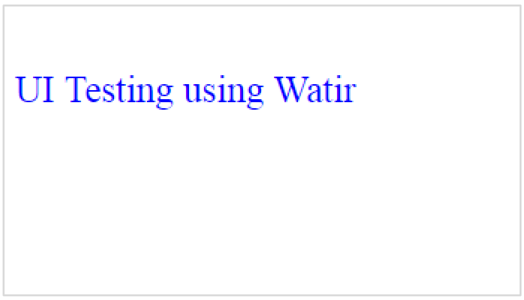


UI Testing using Watir

The Watir code to test the div is shown here:

```
require 'watir'
b = Watir::Browser.new
b.goto('http://localhost/uitesting/div.html')
l = b.div class: 'divtag'
l.exists?
l.text
b.screenshot.save 'divtag.png'
```

Output



UI Testing using Watir

7. Watir — Locating Web Elements

In Watir for testing, you need to locate the elements and it can be done in different ways – by using the id, class or text of the element.

In this chapter, we will see few examples which shows different ways to locate elements.

Using ID of the Element

Test page

```
<html>
<head>
<title>Testing UI using Watir</title>
</head>
<body>
<script type="text/javascript">
function wsentered() {
console.log("inside wsentered");
     var firstname = document.getElementById("firstname");
     if (firstname.value != "") {
            document.getElementById("displayfirstname").innerHTML = "The name
entered is : " + firstname.value;
            document.getElementById("displayfirstname").style.display = "";
     }
}
</script>
<div id="divfirstname">
     Enter First Name : <input type="text" id="firstname" name="firstname"
onchange="wsentered()" />
</div>
<br/>
<br/>
<div style="display:none;" id="displayfirstname">
</div>
</body>
</html>
```

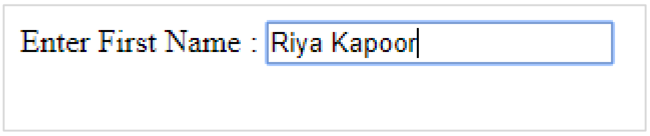
Example

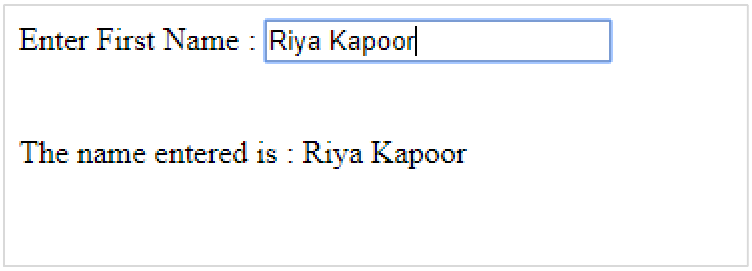
```
require 'watir'
b = Watir::Browser.new :chrome
b.goto('http://localhost/uitesting/textbox.html')
t = b.text_field(id: 'firstname')   // using the id of the textbox  to locate
the textbox
t.exists?
t.set 'Riya Kapoor'
b.screenshot.save 'textboxbefore.png'
t.value
t.fire_event('onchange')
b.screenshot.save 'textboxafter.png'
```

In this example, we are using id of the textbox element to locate it and set the value.

```
t = b.text_field(id: 'firstname')
```

Output


Enter First Name : Riya Kapoor


Enter First Name : Riya Kapoor

The name entered is : Riya Kapoor

In case you need to locate the div, span or any other html tag you can do same using id as follows:

For div

```
browser.div(id: "divid")
browser.div(id: /divid/)
```

For span

```
browser.span(id: "spanid")
browser.span(id: /spanid/)
```

Using NAME of the Element

Test page

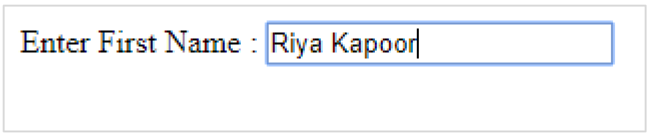
```
<html>
<head>
<title>Testing UI using Watir</title>
</head>
<body>
<script type="text/javascript">
function wsentered() {
console.log("inside wsentered");
     var firstname = document.getElementById("firstname");
     if (firstname.value != "") {
            document.getElementById("displayfirstname").innerHTML = "The name
entered is : " + firstname.value;
            document.getElementById("displayfirstname").style.display = "";
     }
}
</script>
<div id="divfirstname">
     Enter First Name : <input type="text" id="firstname" name="firstname"
onchange="wsentered()" />
</div>
<br/>
<br/>
<div style="display:none;" id="displayfirstname">
</div>
</body>
</html>
```

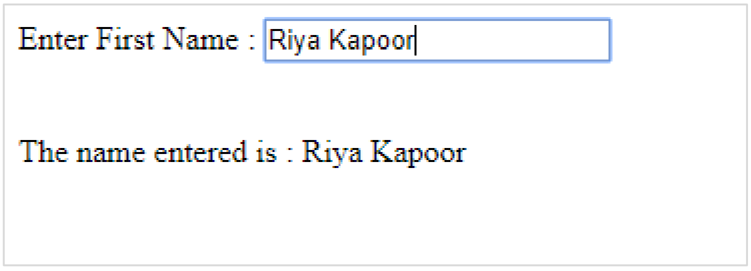
Example

```
require 'watir'
b = Watir::Browser.new :chrome
b.goto('http://localhost/uitesting/textbox.html')
t = b.text_field(name: 'firstname') // name is used to locate the textbox
element
t.exists?
t.set 'Riya Kapoor'
```

```
b.screenshot.save 'textboxbefore.png'
t.value
t.fire_event('onchange')
b.screenshot.save 'textboxafter.png'
```

Output


Enter First Name : Riya Kapoor


Enter First Name : Riya Kapoor

The name entered is : Riya Kapoor

Using Tag Name

You can locate any html elements you want by directly using the html tag as shown below.

For div

```
browser.div(id: "divid")
browser.div(id: /divid/)
```

For span

```
browser.span(id: "spanid")
browser.span(id: /spanid/)
```

For p tag

```
browser.p(id: "ptag")
browser.p(id: /ptag/)
```

For button

```
browser.button(id: "btnid")
browser.button(id: /btnid/)
```

Using Class Name

You can locate the element using its classname. It can be done as shown below:

For div

```
browser.div(class: "divclassname")
browser.div(class: /divclassname/)
```

For span

```
browser.span(class: "spanclassname”)
browser.span(class: /spanclassname/)
```

For p tag

```
browser.p(class: "pclassname")
browser.p(class: /pclassname/)
```

For button

```
browser.button(class: "btnclassname")
browser.button(class: /btnclassname/)
```

For textbox

```
browser.text_field(class: 'txtclassname')
browser.text_field(class: /txtclassname/)
```

You can also pass multiple classes as shown below:

For div

```
browser.div(class: ["class1", "class2"])
```

Using Text

This is yet another way to locate elements by using elements with a text. For example:

```
browser.button(text: "button text")
browser.button(text: /button text/)
```

Using Label

You can use the label of the element to locate it as shown below:

```
browser.text_field(label: "text here"))

browser.text_field(label: /text here/))
```

Using Data Attributes

In-case you have data attributes to your html tags, you can locate the elements using it as shown below:

For example, you can locate the tag as shown below:

```
<div  data-type="test1"></div>
```

You can locate the div as follows:

```
browser.div(data-type: 'test1'))

browser.div(data-type: /test1/))
```

Using Custom Attributes

You can also locate the elements using custom attributes as shown below:

Example of html element

```
<div itemprop=”content”>

 ….

</div>
```

You can locate the div as follows:

```
browser.div(itemprop: ‘content'))

browser.div(itemprop: /content/))
```

Using Visible Attribute

The element using visible attribute can be located as shown below:

```
browser.div(visible: true)

browser.div(visible: false)
```

8. Watir — Working with Iframes

Watir offer easy to use syntax to work with iframes.

Syntax

```
browser.iframe(id: 'myiframe')  // will get the reference of the iframe where
we want to input details.
```

To understand how to deal with iframes and locate the elements inside an iframe, in this chapter, we will work on an example.

Example

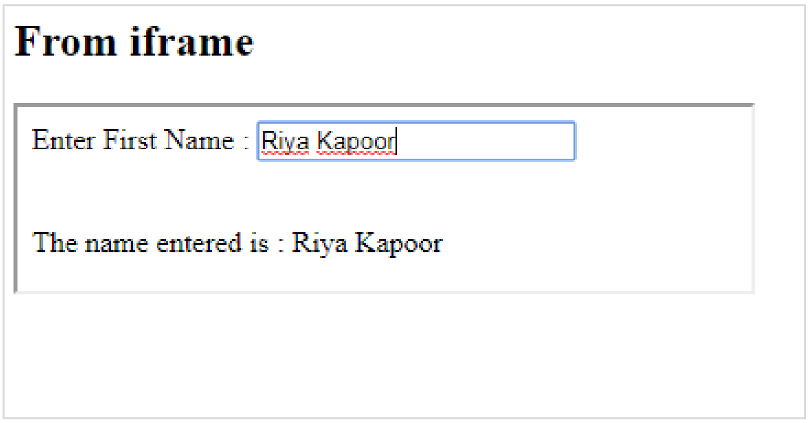
main.html

```
<html>
<head>
<title>Testing using Watir</title>
</head>
<body>
<iframe src="test1.html" id="myiframe" width="500" height="100"></iframe>
</body>
</html>
```

test1.html

```
<html>
<head>
<title>Testing UI using Watir</title>
</head>
<body>
<script type="text/javascript">
function wsentered() {
console.log("inside wsentered");
     var firstname = document.getElementById("firstname");
     if (firstname.value != "") {
            document.getElementById("displayfirstname").innerHTML = "The name
entered is : " + firstname.value;
            document.getElementById("displayfirstname").style.display = "";
     }
}
</script>
<div id="divfirstname">
     Enter First Name : <input type="text" id="firstname" name="firstname"
onchange="wsentered()" />
</div>
<br/>
<br/>
<div style="display:none;" id="displayfirstname">
</div>
</body>
</html>
```

Output



From iframe

Enter First Name : Riya Kapoor

The name entered is : Riya Kapoor

In the above example, the entry form is defined inside an iframe. The Watir code which will help us to locate it and test the form is given below:

Watir Code

```
require 'watir'
b = Watir::Browser.new :chrome
b.goto('http://localhost/uitesting/main.html')
t = b.iframe(id: 'myiframe').text_field
t.set 'Riya Kapoor'
b.screenshot.save 'iframetestbefore.png'
t.fire_event('onchange')
b.screenshot.save 'iframetestafter.png'
```

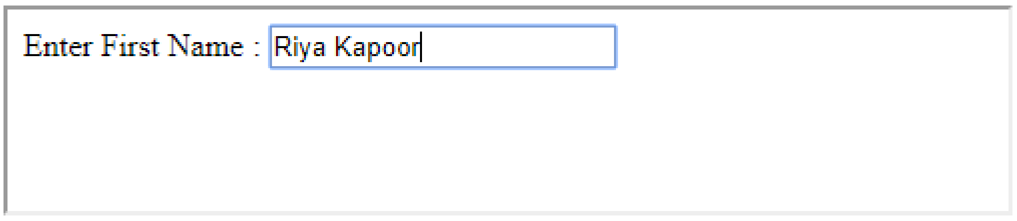
Watir code to locate the iframe in the url given here:

```
t = b.iframe(id: 'myiframe').text_field
```

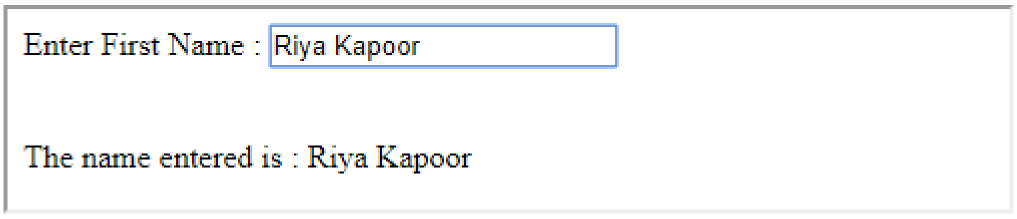
We have used the tag name iframe and the id of the iframe as shown above.

The screenshots of the above code are shown below:

iframetestbefore.png



iframetestafter.png



The name entered is : Riya Kapoor

9. Watir — Automatic Waits

In this chapter, let us understand waits in detail. To understand automatic waits, we have created a simple test page. When user enters text in the textbox onchange event is fired and after 3-seconds the button is enabled.

Watir has a *wait_unit* api call which waits on a particular event or property. We will test the same for the test page as given below:

Syntax

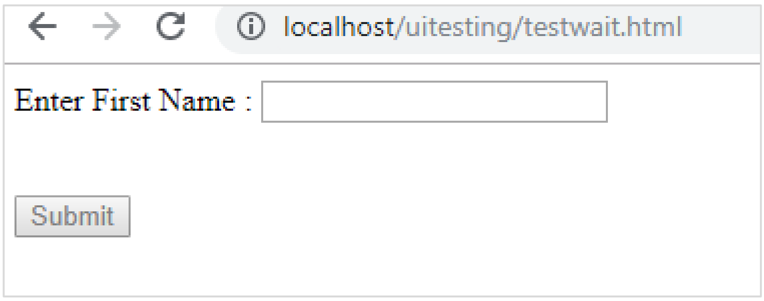
```
browser.button(id: 'btnsubmit').wait_until(&:enabled?) //here the wait is on
the button with id : btnsubmit to be enabled.
```

testwait.html

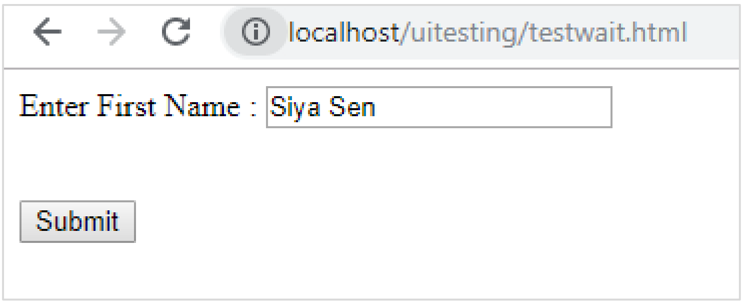
```
<html>
<head>
<title>Testing UI using Watir</title>
</head>
<body>
<script type="text/javascript">
function wsentered() {
     setTimeout(function() {
            document.getElementById("btnsubmit").disabled = false;
     }, 3000);
}

function wsformsubmitted() {
     document.getElementById("showmessage").style.display = "";
}
</script>
<div id="divfirstname">
     Enter First Name : <input type="text" id="firstname" name="firstname"
onchange="wsentered()" />
</div>
<br/>
<br/>
```

```
<button id="btnsubmit" disabled onclick="wsformsubmitted();">Submit</button>
<br/>
<br/>
<div id="showmessage" style="display:none;color:green;font-size:25px;">
Button is clicked
</div>
</body>
</html>
```

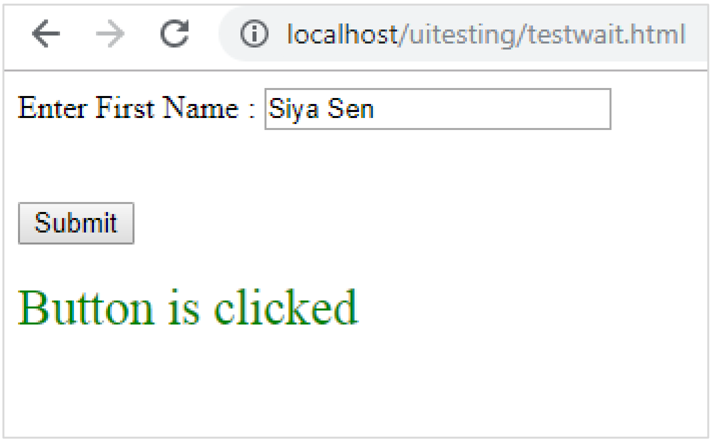
Output



When you enter the text in the textbox, you will have to wait for 3 seconds for the button to be enabled.


localhost/uitesting/testwait.html
Enter First Name : Siya Sen
Submit

When you click the Submit button, the following text is displayed:


localhost/uitesting/testwait.html
Enter First Name : Siya Sen
Submit
Button is clicked

Now since we have added delay for the button to be enabled, it is difficult for the automation to handle such cases. Whenever we have some delay or have to wait on some event or property of the element to be located, we can make use of *wait_until* as shown below:

Watir code using wait_until

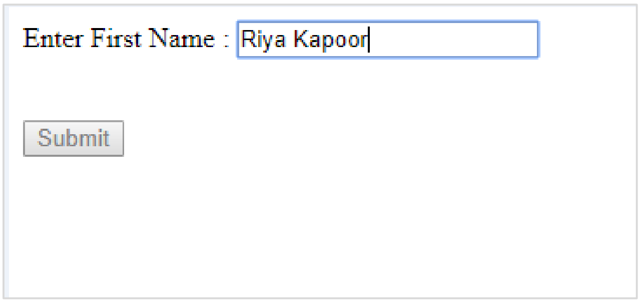
```
require 'watir'
b = Watir::Browser.new :chrome
b.goto('http://localhost/uitesting/testwait.html')
t = b.text_field(name: 'firstname')
t.exists?
t.set 'Riya Kapoor'
b.screenshot.save 'waittestbefore.png'
t.value
t.fire_event('onchange')
btn = b.button(id: 'btnsubmit').wait_until(&:enabled?)
btn.fire_event('onclick');
b.screenshot.save 'waittestafter.png'
```

Next, use the following command

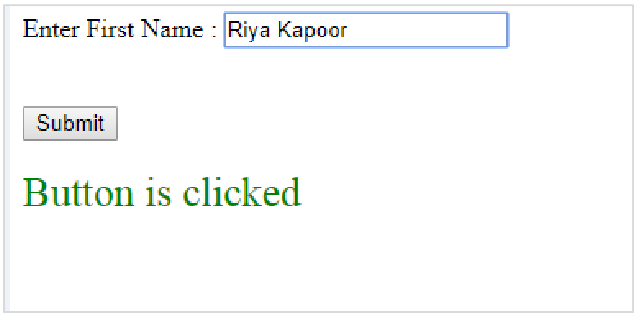
```
btn = b.button(id: 'btnsubmit').wait_until(&:enabled?)
```

Watir is going to wait for the button to get enabled and later go for click event to be fired. The screenshots captured are shown below:

Waittestbefore.png



waittestafter.png



Button is clicked

10. Watir — Headless Testing

In this chapter, we will learn how to use headless option of the Watir webdriver to test the page url.

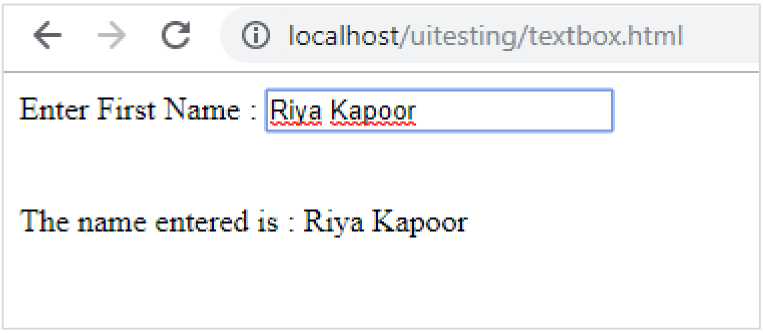
Syntax

```
Browser = Watir::Browser.new :chrome, headless: true
```

The test page that we are going to test is shown here:

```
<html>
<head>
<title>Testing UI using Watir</title>
</head>
<body>
<script type="text/javascript">
function wsentered() {
console.log("inside wsentered");
     var firstname = document.getElementById("firstname");
     if (firstname.value != "") {
            document.getElementById("displayfirstname").innerHTML = "The name
entered is : " + firstname.value;
            document.getElementById("displayfirstname").style.display = "";
     }
}
</script>
<div id="divfirstname">
     Enter First Name : <input type="text" id="firstname" name="firstname"
onchange="wsentered()" />
</div>
<br/>
<br/>
<div style="display:none;" id="displayfirstname">
</div>
</body>
</html>
```

Output


localhost/uitesting/textbox.html

Enter First Name : Riya Kapoor

The name entered is : Riya Kapoor

Watir code

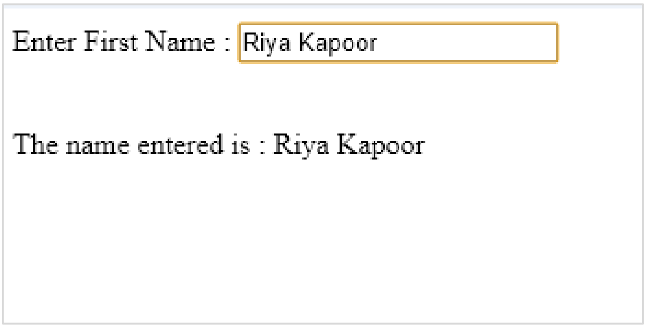
```
require 'watir'
b = Watir::Browser.new :chrome, headless: true
b.goto('http://localhost/uitesting/textbox.html')
t = b.text_field(name: 'firstname')
t.exists?
t.set 'Riya Kapoor'
t.value
t.fire_event('onchange')
b.screenshot.save 'headless.png'
```

We have added the option *headless : true* to the Watir chrome browser. When you execute the Ruby program, it will not open the browser, everything will get executed in the command line:

```
DevTools listening on ws://127.0.0.1:53973/devtools/browser/b4127866-afb8-4c74-
b967-5bacb3354b19
[0505/144843.905:INFO:CONSOLE(8)] "inside wsentered", source:
http://localhost/uitesting/textbox.html (8)
```

We have added console.log message and the same in printed in command line.

The screenshot of headless.png is shown below:


Enter First Name : Riya Kapoor

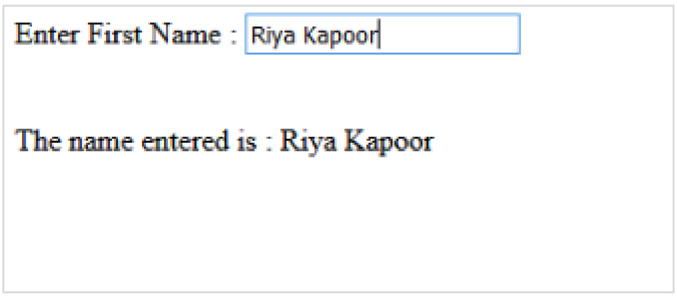
The name entered is : Riya Kapoor

In Firefox

The watir code for Firefox is shown here:

```
require 'watir'
b = Watir::Browser.new :firefox, headless: true
b.goto('http://localhost/uitesting/textbox.html')
t = b.text_field(name: 'firstname')
t.exists?
t.set 'Riya Kapoor'
t.value
t.fire_event('onchange')
b.screenshot.save 'headlessfirefox.png'
```

The screenshot for headlessfirefox.png is shown here:


Enter First Name : Riya Kapoor

The name entered is : Riya Kapoor

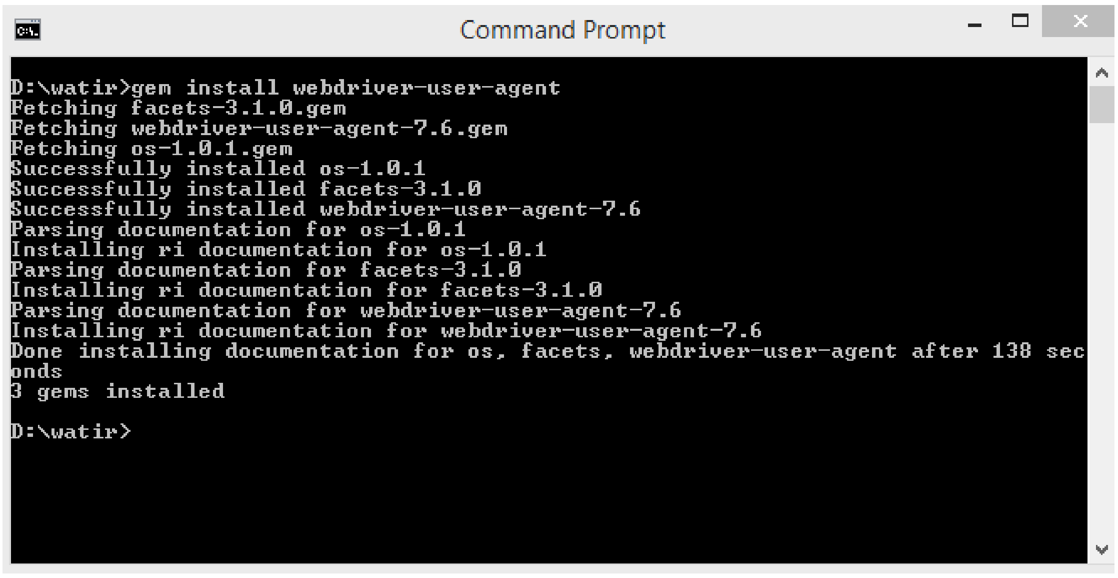
11. Watir — Mobile Testing

For Mobile testing, we are going to use Desktop browser which will act as device browser for testing purpose. Let us understand its procedure in this chapter.

To test your app on mobile browsers we need to install the webdriver-user-agent.

Installing webdriver-user-agent

```
gem install webdriver-user-agent
```



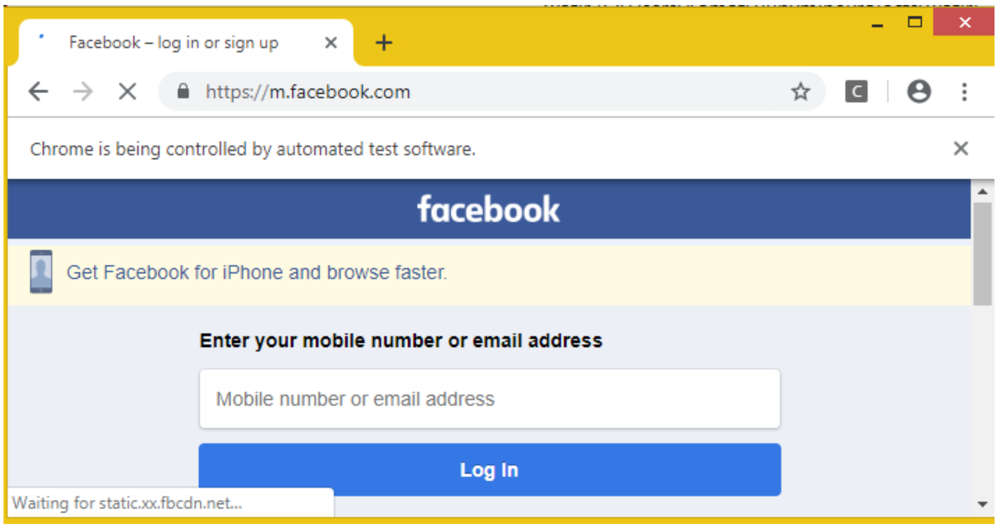
```
D:\watir>gem install webdriver-user-agent
Fetching facets-3.1.0.gem
Fetching webdriver-user-agent-7.6.gem
Fetching os-1.0.1.gem
Successfully installed os-1.0.1
Successfully installed facets-3.1.0
Successfully installed webdriver-user-agent-7.6
Parsing documentation for os-1.0.1
Installing ri documentation for os-1.0.1
Parsing documentation for facets-3.1.0
Installing ri documentation for facets-3.1.0
Parsing documentation for webdriver-user-agent-7.6
Installing ri documentation for webdriver-user-agent-7.6
Done installing documentation for os, facets, webdriver-user-agent after 138 sec
onds
3 gems installed

D:\watir>
```

Now, we are going to use the Webdriver useragent as shown in the example below:

Example

```
require 'watir'

require 'webdriver-user-agent'

driver = Webdriver::UserAgent.driver(browser: :chrome, agent: :iphone,
orientation: :landscape)

browser = Watir::Browser.new driver

browser.goto 'https://facebook.com'

puts "#{browser.url}"

puts browser.url == 'https://m.facebook.com/'
```

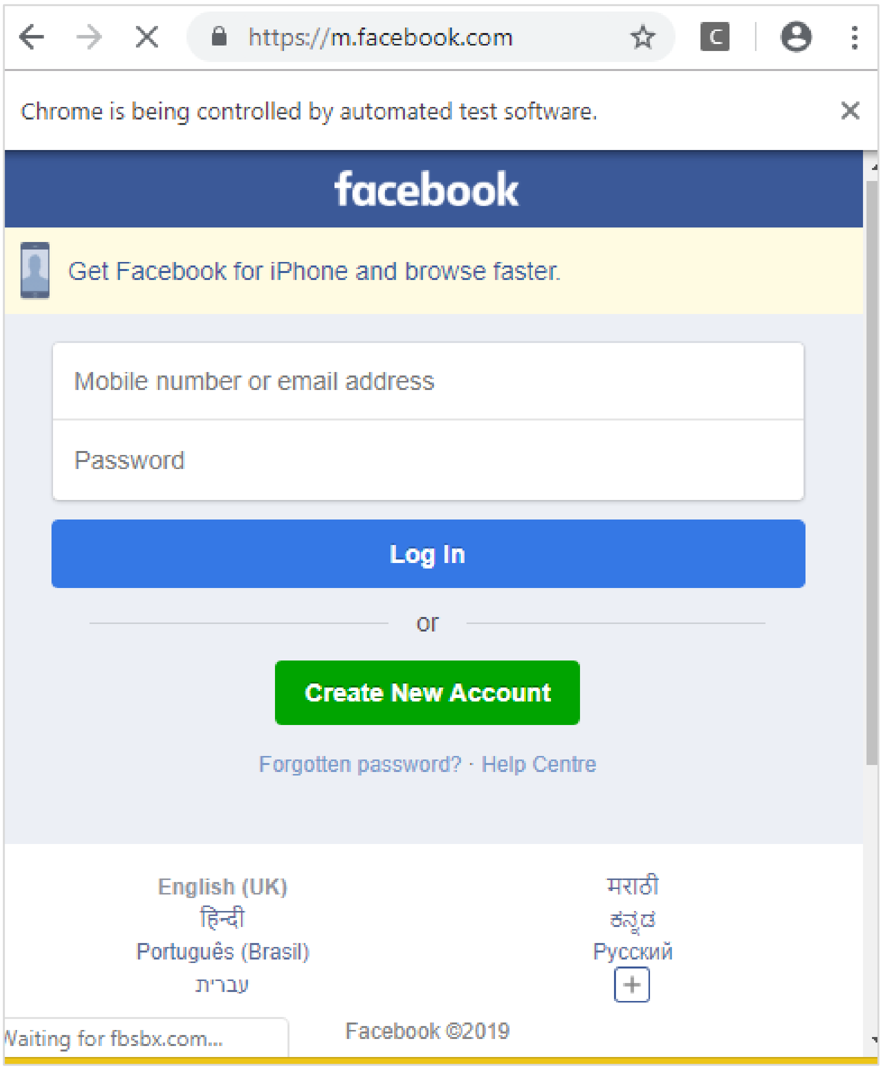
We have given facebook.com url. When you execute it, it opens up in the mobile mode, based on the useragent, as shown below:


Facebook – log in or sign up
https://m.facebook.com
Chrome is being controlled by automated test software.
facebook
Get Facebook for iPhone and browse faster.
Enter your mobile number or email address
Mobile number or email address
Log In
Waiting for static.xx.fbcdn.net...

Let us now try in portrait mode. Use the following code for this purpose:

```
require 'watir'

require 'webdriver-user-agent'

driver = Webdriver::UserAgent.driver(browser: :chrome, agent: :iphone,
orientation: :portrait)

browser = Watir::Browser.new driver

browser.goto 'https://facebook.com'

puts "#{browser.url}"

puts browser.url == 'https://m.facebook.com/'
```

The output in the portrait mode is as shown below:



12. Watir — Capturing Screenshots

Ability to capture screenshots is one of the interesting features available with Watir. During the test automation, you can take screenshots and save the screens. In case, if any error occurs the same can be documented with the help of screenshot.

A simple example along with test page where we have taken the screenshot is discussed below:

Syntax

```
browser.screenshot.save 'nameofimage.png'
```

Test page

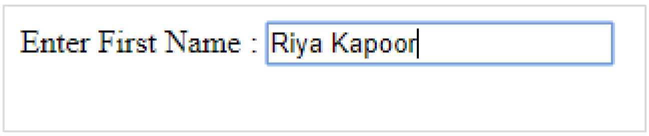
```
<html>
<head>
<title>Testing UI using Watir</title>
</head>
<body>
<script type="text/javascript">
function wsentered() {
console.log("inside wsentered");
     var firstname = document.getElementById("firstname");
     if (firstname.value != "") {
            document.getElementById("displayfirstname").innerHTML = "The name
entered is : " + firstname.value;
            document.getElementById("displayfirstname").style.display = "";
     }}
</script>
<div id="divfirstname">
     Enter First Name : <input type="text" id="firstname" name="firstname"
onchange="wsentered()" />
</div>
<br/>
<br/>
<div style="display:none;" id="displayfirstname">
</div>
</body> </html>
```

Example

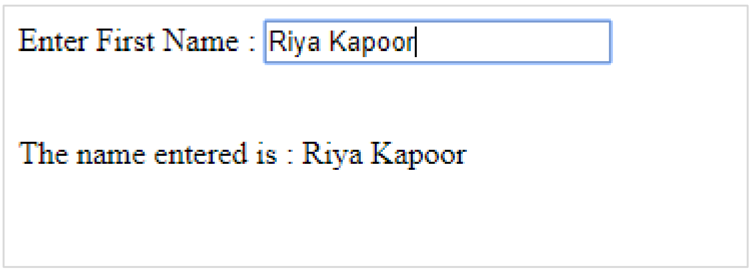
```
require 'watir'
b = Watir::Browser.new :chrome
b.goto('http://localhost/uitesting/textbox.html')
t = b.text_field(id: 'firstname')   // using the id of the textbox  to locate
the textbox
t.exists?
t.set 'Riya Kapoor'
b.screenshot.save 'textboxbefore.png'
t.value
t.fire_event('onchange')
b.screenshot.save 'textboxafter.png'
```

The screenshots we have taken using Watir are shown here:

textboxbefore.png


Enter First Name : Riya Kapoor

textboxafter.png


Enter First Name : Riya Kapoor

The name entered is : Riya Kapoor

13. Watir — Page Objects

Page Object in Watir helps us to reuse the code in the form of classes. Using the page object feature, we can automate our app without having to duplicate any code and also makes the code manageable.

When testing, we can create page object for each of the page we are going to test. Then, we are going to access the methods and properties using the page object.

The reasons behind using page object:

- In case any changes are done to the page at alter changes, re-writing the code is not needed.
- To avoid code redundancy.

We are going to use RSpec to make use of page-object in Watir. Incase if you are not familiar with RSpec, here is a full tutorial available for RSpec for you learning.

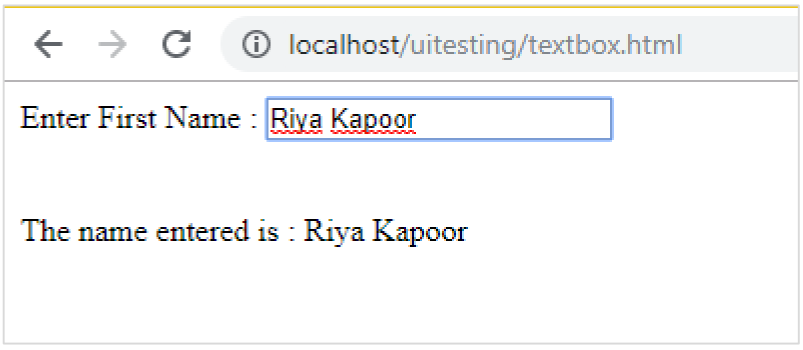
The page that we are going to perform test on is given here:

textbox.html

```
<html>
<head>
<title>Testing UI using Watir</title>
</head>
<body>
<script type="text/javascript">
function wsentered() {
console.log("inside wsentered");
     var firstname = document.getElementById("firstname");
     if (firstname.value != "") {
            document.getElementById("displayfirstname").innerHTML = "The name
entered is : " + firstname.value;
            document.getElementById("displayfirstname").style.display = "";
     }
}
</script>
<div id="divfirstname">
     Enter First Name : <input type="text" id="firstname" name="firstname"
onchange="wsentered()" />
</div>
```

```
<br/>
<br/>
<div style="display:none;" id="displayfirstname">
</div>
</body>
</html>
```

Output



Enter First Name : Riya Kapoor

The name entered is : Riya Kapoor

We will now create page object for the above page as shown below:

pageobjecttest.rb

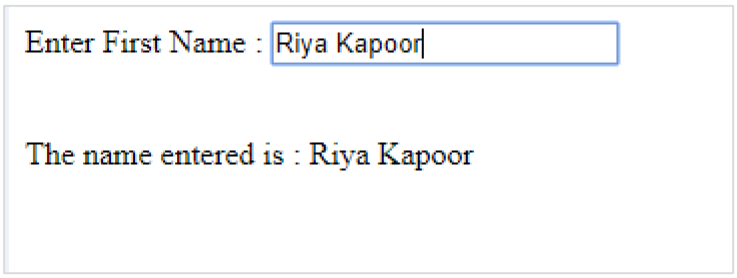
```
class InitializeBrowser
  def initialize(browser)
    @browser = browser
  end
end

class TestPage < InitializeBrowser
  def textbox
    @textbox = TestTextbox.new(@browser)
  end

  def close
    @browser.screenshot.save 'usingpageobject.png'
    @browser.close
  end
end # TestPage

class TestTextbox < InitializeBrowser
```

```
  URL = "http://localhost/uitesting/textbox.html"

  def open
    @browser.goto URL
    self
  end

  def enterdata_as(name)
    name_field.set name
    name_field.fire_event('onchange')
  end

  private
  def name_field
    @browser.text_field(:id => "firstname")
  end
end # TestTextbox
```

There are three classes defined - InitializeBrowser, TestPage and TestTextbox:

- **InitializeBrowser**: This will initialize the browser opened and share the browser object with TestPage and TestTextbox classes.
- **TestPage**: This class will have object reference to TestTextbox and contains the method to capture screenshot and close the browser.
- **TestTextbox**: This class will have methods to open the page url, give reference to textfield , set the data and fire onchange event.

Once you execute the code shown above, you can see the output as shown below:


Enter First Name : Riya Kapoor

The name entered is : Riya Kapoor

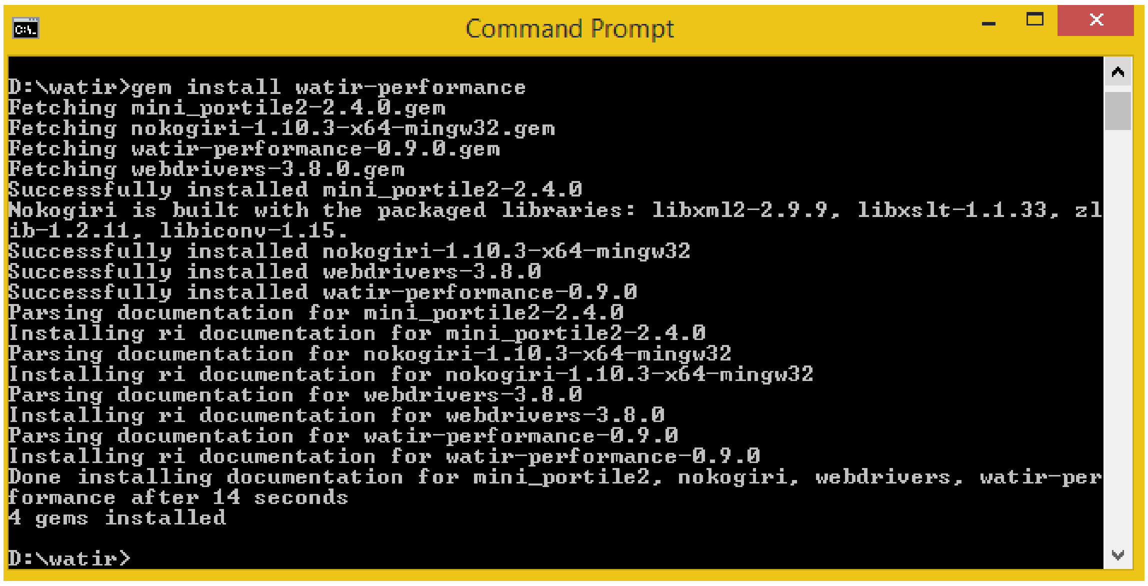
14. Watir — Page Performance

Watir Page performance feature allows you to track the response time metrics and it works fine in Chrome, Firefox, IE9 and above. Safari browser does not have the support as of now.

Let us take a closer look on how to use this feature. To make use of it, we need to install watir-performance using gem as shown below:

Command

```
gem install watir-performance
```



```
D:\watir>gem install watir-performance
Fetching mini_portile2-2.4.0.gem
Fetching nokogiri-1.10.3-x64-mingw32.gem
Fetching watir-performance-0.9.0.gem
Fetching webdrivers-3.8.0.gem
Successfully installed mini_portile2-2.4.0
Nokogiri is built with the packaged libraries: libxml2-2.9.9, libxslt-1.1.33, zl
ib-1.2.11, libiconv-1.15.
Successfully installed nokogiri-1.10.3-x64-mingw32
Successfully installed webdrivers-3.8.0
Successfully installed watir-performance-0.9.0
Parsing documentation for mini_portile2-2.4.0
Installing ri documentation for mini_portile2-2.4.0
Parsing documentation for nokogiri-1.10.3-x64-mingw32
Installing ri documentation for nokogiri-1.10.3-x64-mingw32
Parsing documentation for webdrivers-3.8.0
Installing ri documentation for webdrivers-3.8.0
Parsing documentation for watir-performance-0.9.0
Installing ri documentation for watir-performance-0.9.0
Done installing documentation for mini_portile2, nokogiri, webdrivers, watir-per
formance after 14 seconds
4 gems installed

D:\watir>
```

We are done with installing watir-performance. The metrics that are supported are:

- summary
- navigation
- memory
- Timing

A working example using watir-performance is discussed here. Here, we will check the response time for site:https://www/tutorialspoint.com as shown below:

```
require 'watir'

require 'watir-performance'

10.times do
```

```
  b = Watir::Browser.new :chrome
  b.goto 'https://www.tutorialspoint.com'
  load_secs = b.performance.summary[:response_time] / 1000
  puts "Load Time: #{load_secs} seconds."
  b.close
end
```

Output

```
Load Time: 7 seconds.
Load Time: 7 seconds.
Load Time: 5 seconds.
Load Time: 5 seconds.
Load Time: 6 seconds.
Load Time: 5 seconds.
Load Time: 5 seconds.
Load Time: 13 seconds.
Load Time: 12 seconds.
Load Time: 5 seconds.
```

Using performance.timing

```
require 'watir'
require 'watir-performance'

b = Watir::Browser.new :chrome
b.goto 'https://www.tutorialspoint.com'
load_secs = b.performance.timing[:response_end] -
b.performance.timing[:response_start]
puts "Time taken to respond is  #{load_secs} seconds."
b.close
```

Output

```
Time taken to respond is  41 seconds.
```

Using performance.navigation

```
require 'watir'
require 'watir-performance'

b = Watir::Browser.new :chrome
b.goto 'https://www.tutorialspoint.com'
perf_nav = b.performance.navigation
puts "#{perf_nav}"
b.close
```

Output

```
{:type_back_forward=>2, :type_navigate=>0, :type_reload=>1,
:type_reserved=>255, :redirect_count=>0, :to_json=>{}, :type=>0}
```

Using performance.memory

```
require 'watir'
require 'watir-performance'

b = Watir::Browser.new :chrome
b.goto 'https://www.tutorialspoint.com'
memory_used = b.performance.memory
puts "#{memory_used}"
b.close
```

Output

```
{:js_heap_size_limit=>2136997888, :total_js_heap_size=>12990756,
:used_js_heap_size=>7127092}
```

15. Watir — Cookies

In this chapter, we will learn how to work with cookies using Watir.

A simple example that will fetch the cookies for a URL given is discussed here.

Syntax to fetch cookies

```
browser.cookies.to_a
```

Example

```
require 'watir'


b = Watir::Browser.new :chrome

b.goto 'https://www.tutorialspoint.com'

puts b.cookies.to_a
```

Output

```
{:name=>"_gat_gtag_UA_232293_6", :value=>"1", :path=>"/",
:domain=>".tutorialspoint.com", :expires=>2019-05-03 08:33:58 +0000,
:secure=>false}

{:name=>"_gid", :value=>"GA1.2.282573155.1556872379", :path=>"/",
:domain=>".tutorialspoint.com", :expires=>2019-05-04 08:32:57 +0000,
:secure=>false}

{:name=>"_ga", :value=>"GA1.2.2087825339.1556872379", :path=>"/",
:domain=>".tutorialspoint.com", :expires=>2021-05-02 08:32:57 +0000,
:secure=>false}
```

Now let us add cookies as shown below:

Syntax to add cookies

```
browser.cookies.add 'cookiename', 'cookievalue', path: '/', expires: (Time.now
+ 10000), secure: true
```

Example

```
require 'watir'
b = Watir::Browser.new :chrome
b.goto 'https://www.tutorialspoint.com'
puts b.cookies.to_a
b.cookies.add 'cookie1', 'testing_cookie', path: '/', expires: (Time.now +
10000), secure: true
puts b.cookies.to_a
```

Output Before Adding cookie

```
{:name=>"_gat_gtag_UA_232293_6", :value=>"1", :path=>"/",
:domain=>".tutorialspoint.com", :expires=>2019-05-03 08:44:23 +0000,
:secure=>false}
{:name=>"_gid", :value=>"GA1.2.1541488984.1556873004", :path=>"/",
:domain=>".tutorialspoint.com", :expires=>2019-05-04 08:43:24 +0000,
:secure=>false}
{:name=>"_ga", :value=>"GA1.2.1236163943.1556873004", :path=>"/",
:domain=>".tutorialspoint.com", :expires=>2021-05-02 08:43:24 +0000,
:secure=>false}
```

Output After adding cookie

```
{:name=>"_gat_gtag_UA_232293_6", :value=>"1", :path=>"/",
:domain=>".tutorialspoint.com", :expires=>2019-05-03 08:44:23 +0000,
:secure=>false}
{:name=>"_gid", :value=>"GA1.2.1541488984.1556873004", :path=>"/",
:domain=>".tutorialspoint.com", :expires=>2019-05-04 08:43:24 +0000,
:secure=>false}
{:name=>"_ga", :value=>"GA1.2.1236163943.1556873004", :path=>"/",
:domain=>".tutorialspoint.com", :expires=>2021-05-02 08:43:24 +0000,
:secure=>false}
{:name=>"cookie1", :value=>"testing_cookie", :path=>"/",
:domain=>"www.tutorialspoint.com", :expires=>2039-04-28 08:43:35 +0000,
:secure=>true}
```

Note that the last one is the one we added using watir.

Clear Cookies

Syntax

```
browser.cookies.clear
```

Example

```
require 'watir'


b = Watir::Browser.new :chrome

b.goto 'https://www.tutorialspoint.com'

puts b.cookies.to_a

b.cookies.clear

puts b.cookies.to_a
```

Output

```
{:name=>"_gat_gtag_UA_232293_6", :value=>"1", :path=>"/",
:domain=>".tutorialspoint.com", :expires=>2019-05-03 08:48:29 +0000,
:secure=>false}

{:name=>"_gid", :value=>"GA1.2.1264249563.1556873251", :path=>"/",
:domain=>".tutorialspoint.com", :expires=>2019-05-04 08:47:30 +0000,
:secure=>false}

{:name=>"_ga", :value=>"GA1.2.1001488637.1556873251", :path=>"/",
:domain=>".tutorialspoint.com", :expires=>2021-05-02 08:47:30 +0000,
:secure=>false


Empty response ie a blank line will get printed after cookie.clear is called.
```

Delete a particular cookie

Syntax

```
browser.cookies.delete 'nameofthecookie'
```

Example

```
require 'watir'
b = Watir::Browser.new :chrome
b.goto 'https://www.tutorialspoint.com'
puts b.cookies.to_a
puts b.cookies.delete "_ga"
puts b.cookies.to_a
```

Output

```
All cookies:
{:name=>"_gat_gtag_UA_232293_6", :value=>"1", :path=>"/",
:domain=>".tutorialspoint.com", :expires=>2019-05-03 08:52:38 +0000,
:secure=>false}
{:name=>"_gid", :value=>"GA1.2.1385195240.1556873499", :path=>"/",
:domain=>".tutorialspoint.com", :expires=>2019-05-04 08:51:37 +0000,
:secure=>false}
{:name=>"_ga", :value=>"GA1.2.1383421835.1556873499", :path=>"/",
:domain=>".tutorialspoint.com", :expires=>2021-05-02 08:51:37 +0000,
:secure=>false}


After delete cookie with name _ga
{:name=>"_gat_gtag_UA_232293_6", :value=>"1", :path=>"/",
:domain=>".tutorialspoint.com", :expires=>2019-05-03 08:52:38 +0000,
:secure=>false}
{:name=>"_gid", :value=>"GA1.2.1385195240.1556873499", :path=>"/",
:domain=>".tutorialspoint.com", :expires=>2019-05-04 08:51:37 +0000,
:secure=>false}
```

16. Watir — Proxies

Watir allows to use proxy with the help of proxy object which needs to be used with the browser.

Syntax

```
proxy = {
    http: '127.0.0.1:8080',
    ssl:  '127.0.0.1:8080'
}

b = Watir::Browser.new :chrome, proxy: proxy
```

An example on how to use proxy with Chrome browser is shown below:

Example

```
require "watir"
proxy = {
    http: '127.0.0.1:8080',
    ssl:  '127.0.0.1:8080'
}

b = Watir::Browser.new :chrome, proxy: proxy
b.goto 'google.com'
b.screenshot.save 'proxy.png'
```

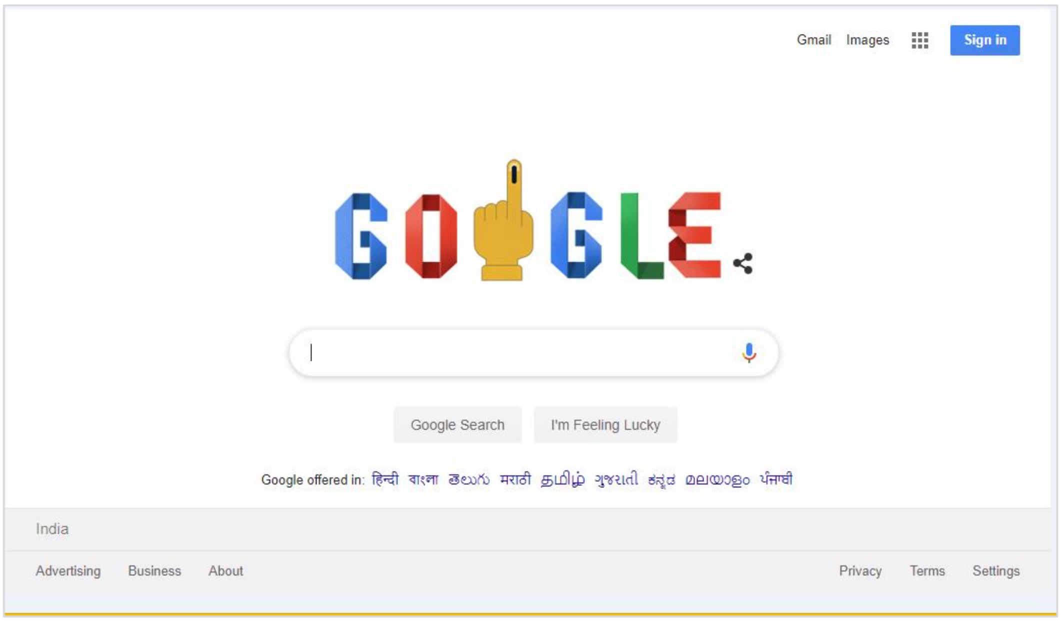
We have used proxy object as shown below:

```
proxy = {
    http: '127.0.0.1:8080',
    ssl:  '127.0.0.1:8080'
}
```

The proxy address details are to be used for both http and ssl. We can use proxy with chrome browser as shown below:

```
b = Watir::Browser.new :chrome, proxy: proxy
```

The output proxy.png is shown below:



An example on how to use proxy with Firefox browser is discussed below:

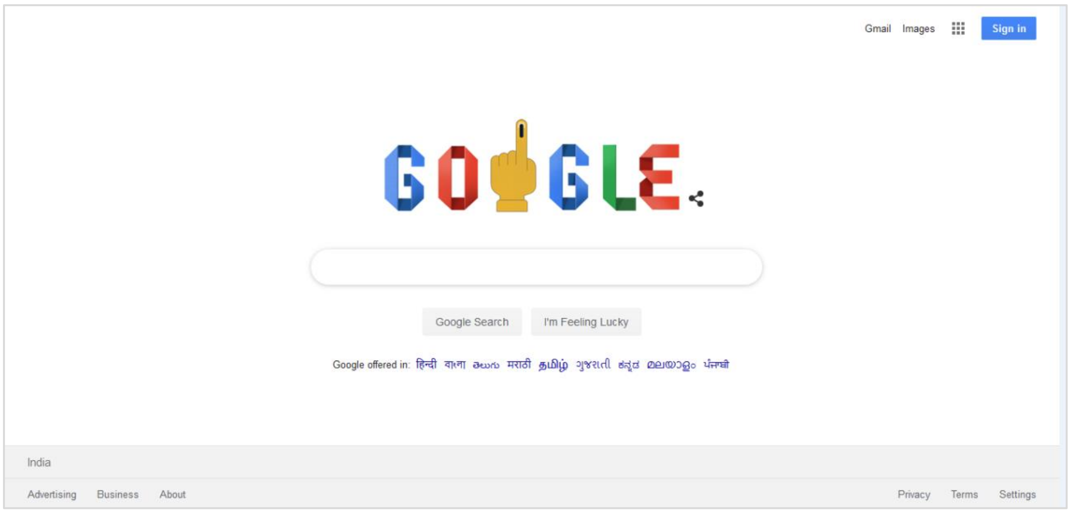
Example

```
require "watir"
proxy = {
    http: '127.0.0.1:8080',
    ssl:  '127.0.0.1:8080'
}

b = Watir::Browser.new :firefox, proxy: proxy
b.goto 'google.com'
b.screenshot.save 'proxyfirefox.png'
```

You can add the proxy details as shown below:

```
proxy = {
    http: '127.0.0.1:8080',
    ssl:  '127.0.0.1:8080'
}
b = Watir::Browser.new :firefox, proxy: proxy
```

The output proxyfirefox.png is shown here:



17. Watir — Alerts

In this chapter we will understand how to handle alerts using Watir.

Syntax

```
browser.alert.exists?
browser.alert.ok
browser.alert.close
```

Testpage

```
<html>
<head>
<title>Testing Alerts Using Watir</title>
</head>
<body>
<script type="text/javascript">
function wsformsubmitted() {
     alert("Button is Clicked !");
}
</script>
<button id="btnsubmit" onclick="wsformsubmitted();">Submit</button>
</body>
</html>
```

Watir Code

```
require 'watir'
b = Watir::Browser.new :chrome
b.goto('http://localhost/uitesting/testalert.html')
b.button(id: 'btnsubmit').click
b.alert.ok
b.screenshot.save 'alerttest.png'
```

The output alerttest.png is shown here:

18. Watir — Downloads

We have buttons or links in the UI or our website which downloads a pdf, or a doc. We can test that for using with Watir by giving some preferences to the browser.

The syntax for downloading:

```
prefs = {
    'download' => {
        'default_directory' => "C:/download",
        'prompt_for_download' => false,
    }
}
b = Watir::Browser.new :chrome, options: {prefs: prefs}
```

The prefs has download wherein we need to give the path where we want the file to be stored after download and the same has to be given to the browsers using options as shown in the syntax above.

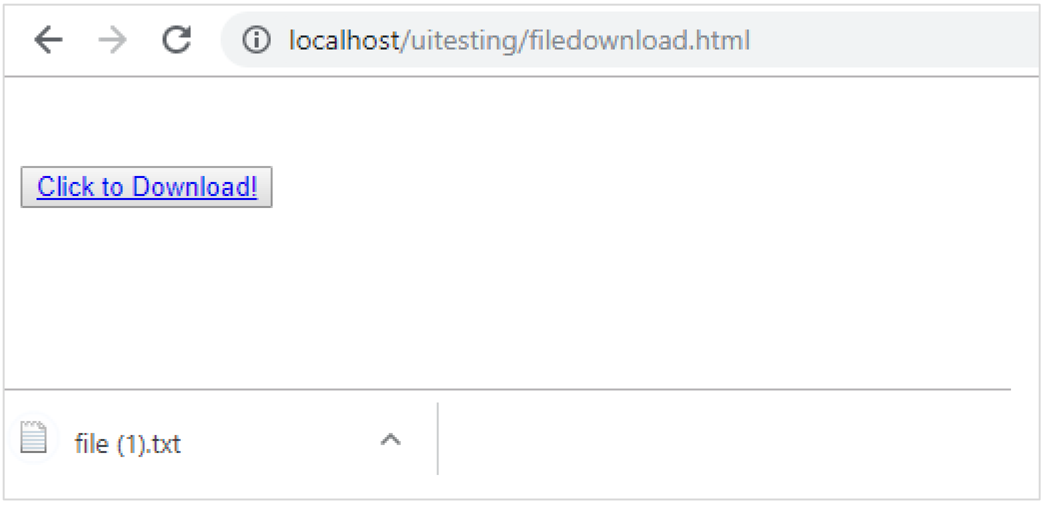
A working example is shown here. Here, we have created test page with a button, which when clicked will download a file called file.txt as shown below:

```
<html>
<head>
<title>Testing UI using Watir</title>
</head>
<body>
<br/>
<br/>
<button id="btnsubmit">
    <a href="file.txt" download>Click to Download!</a>
</button>
<br/>
</body>
</html>
```

file.txt

```
This is for testing watir download
```

Output

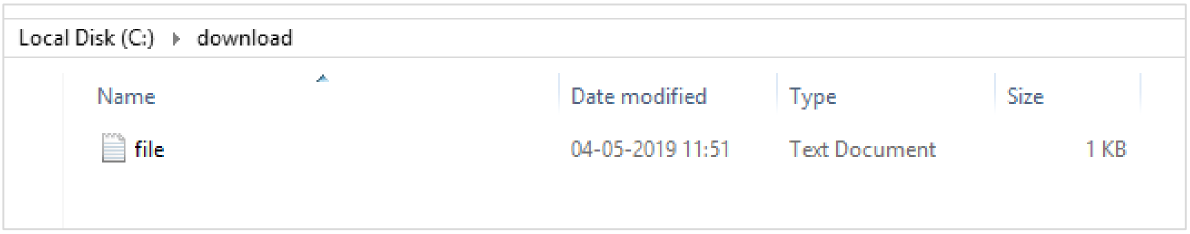


When you click the download button, the file is downloaded.

Now let us test the same using Watir:

```
require 'watir'
prefs = {
    'download' => {
        'default_directory' => "C:/download",
        'prompt_for_download' => false,
    }
}
b = Watir::Browser.new :chrome, options: {prefs: prefs}
b.goto('http://localhost/uitesting/filedownload.html')
b.button(id: 'btnsubmit').click
b.screenshot.save 'testdownload.png'
```

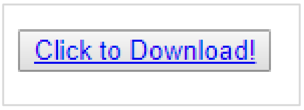
The path we have given to store the downloaded file is "C:/download". When we execute above code we will have the file download in download path given as shown below:



Local Disk (C:) ▸ download

Name	Date modified	Type	Size
file	04-05-2019 11:51	Text Document	1 KB

The output testdownload.png is as shown here:



19. Watir — Browser Windows

You will come across cases where we have to use popup window or opening of a new browser window. In this chapter, we will discuss how to test such cases using Watir.

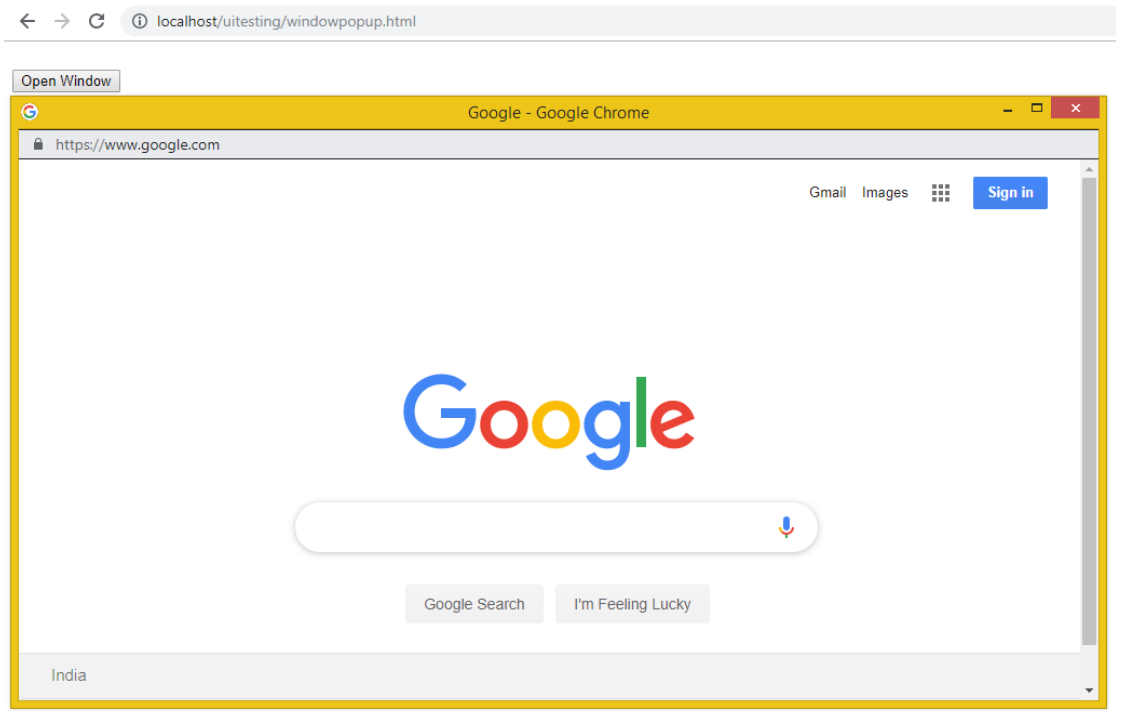
Syntax

```
browser.window
```

A working example that we are going to test is given here:

```
<html>
<head>
<title>Testing UI using Watir</title>
</head>
<body>
<script type="text/javascript">
function wsclick() {
     var myWindow = window.open("https://www.google.com/", "mywindow",
"width=1000,height=500");
}
</script>
<form name="myform" method="POST">
<div><br>
<input type="button" id="btnsubmit" name="btnsubmit"
value="submit"  onclick="wsclick()"/>
<br>
</div>
</form>
<br/>
</body>
</html>
```

Output



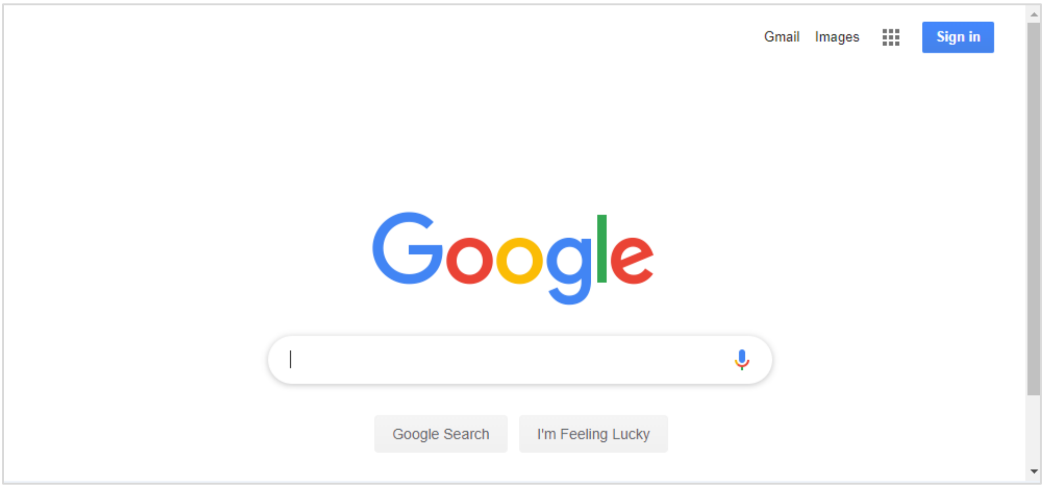
On-click of button *Open Window*, the popup window opens up. Here, the url we have given is https://www.google.com. Now let us test the same using Watir/

Example

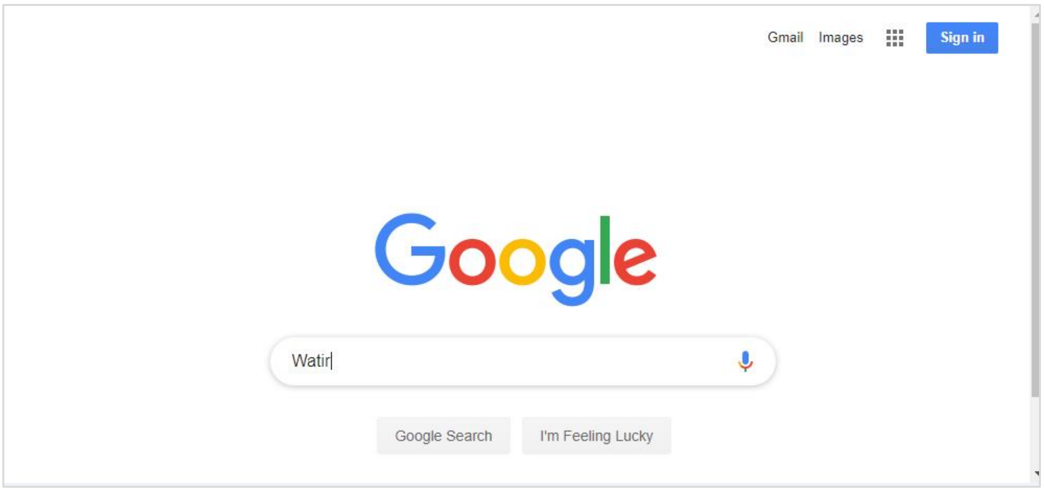
```
require 'watir'
b = Watir::Browser.new :chrome
b.goto('http://localhost/uitesting/windowpopup.html')
b.button(id: 'btnsubmit').click
b.window(title: 'Google').use do
  b.screenshot.save 'popupwindow.png'
  t = b.text_field(class: 'gLFyf')
  t.set 'Watir'
  b.screenshot.save 'popupwindowbefore.png'
  b.button(name: 'btnK').click
  b.screenshot.save 'popupwindowafter.png'
end
```

The screenshots that we have taken are given below:

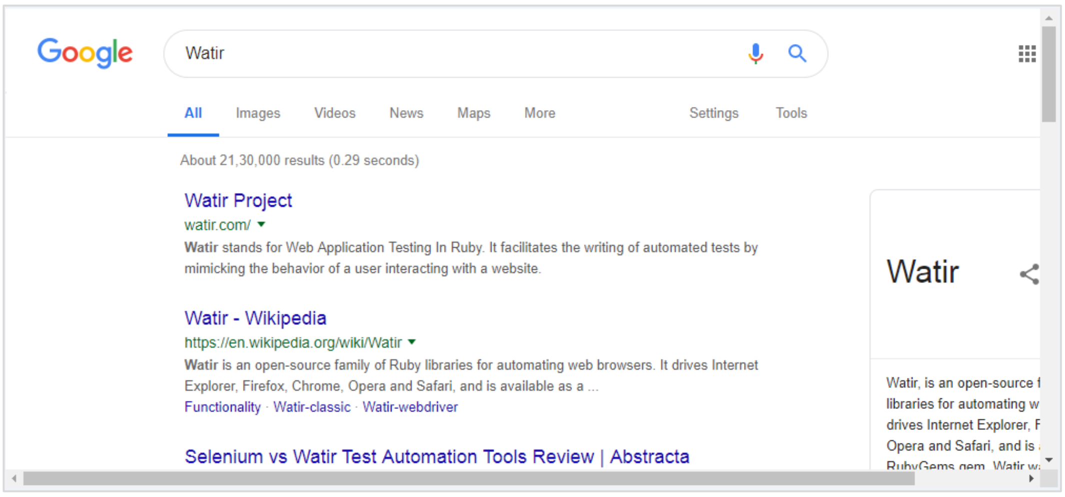
popupwindow.png



popupwindowbefore.png



popupwindowafter.png



About 21,30,000 results (0.29 seconds)

Watir Project
watir.com/
Watir stands for Web Application Testing In Ruby. It facilitates the writing of automated tests by mimicking the behavior of a user interacting with a website.

Watir - Wikipedia
https://en.wikipedia.org/wiki/Watir
Watir is an open-source family of Ruby libraries for automating web browsers. It drives Internet Explorer, Firefox, Chrome, Opera and Safari, and is available as a ...
Functionality · Watir-classic · Watir-webdriver

Selenium vs Watir Test Automation Tools Review | Abstracta

Watir

Watir, is an open-source f
libraries for automating w
drives Internet Explorer, F
Opera and Safari, and is
RubyGems gem. Watir wa