Watir 6 FAQ

Frequently Asked Questions

Answers

What about watir-webdriver?

All of the watir-webdriver code has been moved into the watir gem. The future of Watir is using the W3C specification for browser automation, and that means basing the active implementation of Watir on Selenium.

Why are my tests failing because of a chromedriver error?

Due to the changes Mozilla has made recently, it makes more sense for Chrome to be the default browser. Download ChromeDriver and place it somewhere on your PATH. See here for more information.

Why are my tests failing because of a geckodriver error?

Mozilla has updated Firefox to no longer allow the previous implementation of Firefox Driver to function. The replacement implementation is geckodriver. Download geckodriver and place it somewhere on your PATH. See here for more information.

Why are my Internet Explorer tests failing?

Watir now relies on the Selenium driver for Internet Explorer. If you are having issues updating your tests to get them to pass, please ask us for help.

Why am I getting warnings about #always_locate and/or #prefer_css?

Based on recent changes, neither of these options added significant additional functionality so they have been removed. Read this post for more information.

Why am I getting warnings about #when_present being deprecated?

Watir by default now automatically calls `#when_present` and `#when_enabled` before taking actions on elements as appropriate, so their use is redundant. If you are using the default settings you can just delete these methods from your tests. If you have decided to change away from the recommended settings (by explicitly setting `Watir.relaxed_locate = false`), then swap out from these:

```
browser.element.when_present.click
browser.element.when_enabled.click
```

to these:

```
browser.element.wait_until(&:present?).click
browser.element.wait_until(&:enabled?).click
```

Why am I getting warnings about keywords versus arguments?

In order to make it more explicit what the method is doing, `#wait_until` and `#wait_while` methods now accept keyword arguments, and the use of ordered parameters is deprecated. Previously, you would do this:

```
# Specify both parameters:
element.wait_until(5, "Oops not not there") { |el| el.present? }
# Specify only timeout:
element.wait_until(5) { |el| el.present? }
# Specify only error message:
element.wait_until(nil, "Oops not not there") { |el| el.present?
```

Whereas now you need to explicitly specify the timeout and/or the message keywords:

```
# Specify both parameters:
element.wait_until(timeout: 5, message: "Oops not not there") {
# Specify only timeout:
element.wait_until(timeout: 5) { |el| el.present? }
# Specify only error message:
element.wait_until(message: "Oops not not there") { |el| el.pres
```

Why are my tests taking so long?

If you are seeing an error like: “this code has slept for the duration of the default timeout” then you have an issue with the new automatic waits. The most likely cause is that you are taking an action on an element that is not there and rescuing the exception, like this:

```
begin
  browser.element.click
  take_action_if_present
rescue Watir::Exception::UnknownObjectException
  take_action_if_not_present
end
```

Ideally this gets replaced with:

```
  element = browser.element
  if element.present?
    take_action_if_present
  else
    take_action_if_not_present
  end
```

The other alternative is to set

```
Watir.relaxed_locate = false
```

What is with the "&:" symbols in the new wait documentation?

This notation makes use of one of Ruby's more powerful features with regards to blocks (also known as closures). The ampersand ("&") attempts to convert whatever object it is in front of into a block. In the case of this:

```
browser.div(id: 'foo').wait_until(&:present?)
```

The object is a symbol (`:present?`), and symbols have a `#to_proc` method. This method makes the above code equivalent to this:

```
browser.div(id: 'foo').wait_until { |element| element.present? }
```

It's a powerful way to abbreviate unnecessarily long lines of code. There are plenty of tutorials and descriptions of Ruby's `#to_proc` methods if you'd like to learn more.

What else is new?

Take a look at the changelog for all of the updates, but a list of new features:

1. Collections do not go stale on DOM refresh
2. Elements can be located with a visible filter
3. Error messages include more specific details about elements that aren't found
4. Watir automatically waits for elements to be ready before taking actions on them
5. The `text_field` method can now be used to locate most HTML5 inputs

---
