Extracted from:

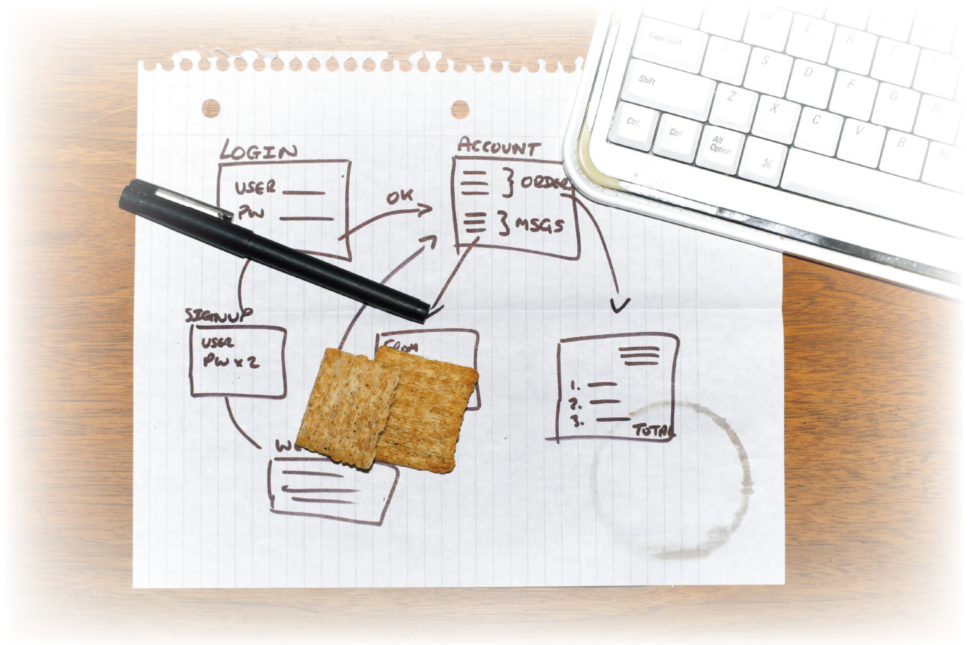
Scripted GUI Testing with Ruby

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina

The Pragmatic Programmers

Scripted GUI Testing with Ruby



Ian Dees

Edited by Jacquelyn Carter

The Facets of Ruby Series

Printed in the United States of America.
ISBN-13: 978-1-9343561-8-0
Printed on acid-free paper.
Book version: P1.1—January 2012

9.4 Interacting with Ajax

So far, we've looked only at straight HTML pages. How do we test something a little more interactive, like a JavaScript-heavy site?

For this section, we'll use a simple drag-and-drop list. I've included one in the source code for this book, in dragdrop.html. To use it, you'll need an open source JavaScript library called script.aculo.us.[10]

Download and open the latest source archive from script.aculo.us site. Copy all the .js files from the lib and src directories to the same folder where you're keeping dragdrop.html.

Open dragdrop.html manually in your browser, and click the "Reorder" link. If the JavaScript files are in the right place, then you should see a bunch of little black draggable handles, as in Figure 11, *Drag and drop*, on page ?.

Selenium works best when the browser is going through an actual web server, rather than just reading files off a disk. For the tests in this section, fire up a separate command prompt, and run the trivial Ruby-based server included in this chapter's source code (web_server.rb).[11] I'm assuming you're running behind a firewall or taking some other measure to keep people from hitting this page from the outside world.

Let's write a couple of tests to exercise the drag-and-drop capabilities of the joke list. First, here's the outline of the RSpec description:

```
require 'joke_list'
describe JokeList do
  before do
    @list = JokeList.new
  end
  after do
    @list.close
  end
  # tests will go here...
end
```

For the first example, we'll just do a single drag to the end of the list:

Download tubes/list_spec.rb

```
it 'lets me drag an item to the end' do
  @list.order('doctor').should == 2
  @list.move 2, 5
```

10. http://script.aculo.us
11. Or you can use your favorite web server package and adjust the port numbers in the script if you want to use something other than 8000.

```
  @list.order('doctor').should == 5
end
```

Let's go ahead and fill in enough of the JokeList class to drive the web browser for that example.

Here's the setup and teardown code:

Download tubes/joke_list.rb

```
require 'rubygems'
require 'selenium'

class JokeList
  def initialize
    @browser = Selenium::SeleniumDriver.new \
      'localhost', 4444, '*firefox', "http://localhost:8000", 10000

    @browser.start
    @browser.open 'http://localhost:8000/dragdrop.html'
  end

  def close
    @browser.stop
  end
end
```

JokeList also needs an order() method so we can see where a given joke is in the list. I've given each joke a unique id attribute in the HTML, and Selenium's get_element_index() method will take those IDs directly:

Download tubes/joke_list.rb

```
class JokeList
  def order(item)
    @browser.get_element_index(item).to_i + 1
  end
end
```

Now we need to add drag and drop. There are a few different ways to do this in Selenium. If we're just doing something really simple like moving an item past the end of a list, we can say this...

```
@browser.drag_and_drop element, '0, +300'
```

which will break as soon as we try to test a list that's taller than 300 pixels. Coordinates retrieved at runtime are much more resilient than hard-coded offsets:

```
last_y = @browser.get_element_position_top(last_element) +
         @browser.get_element_height(last_element)
@browser.drag_and_drop element, "0, #{last_y}"
```

But it turns out Selenium lets us specify the drop target directly:

```
@browser.drag_and_drop_to_object element, last_element
```

Here's what it looks like in context:

Download tubes/joke_list.rb

```
class JokeList
  Reorder = '//a[@id="reorder"]'
① Draggable = 'selenium.browserbot.findElement("css=.drag").visible()'
  Locked = '!' + Draggable

  def move(from_order, to_order)
    from_element = "//li[#{from_order}]/span[@class='drag']"
    to_element   = "//li[#{to_order}]/span[@class='drag']"

    @browser.click Reorder
②   @browser.wait_for_condition Draggable, 2000

    @browser.drag_and_drop_to_object from_element, to_element

    @browser.click Reorder
③   @browser.wait_for_condition Locked, 2000
  end
end
```

One thing to note is that XPath uses 1 to denote the first item in a list, rather than the 0 we're used to from Ruby. To keep things straight, I'm using order or pos for XPath-style, 1-based positions, and index for Ruby-style, 0-based indices.

Another thing we need to worry about is timing. When you're testing an Ajax page, you often need to wait for a portion of a page to refresh. To simulate a server round-trip, dragdrop.html pauses slightly before showing the drag handles when you click the "Reorder" link.

A naïve approach would be to add a fixed delay to our test script. But those are awfully prone to breakage. Instead, we're using Selenium's handy wait_for_condition() method at ② and ③. This function will wait until a given JavaScript piece evaluates to true. To access elements on the page Selenium is controlling, you go through the browserbot attribute, like we're doing at ①.

We'll write one more example—something a little more substantial—and move on. Just for fun, let's implement an alphabetic sort on the list. An end user might do something like an insertion sort: visually scan the list for the item that should go last, move it to the end, scan for the item that should be next-to-last, and so forth.[12]

Download tubes/list_spec.rb
```
it 'lets me drag multiple items to sort' do
  original = @list.items

  original.length.downto(1) do |last_pos|
    subset = @list.items[0..last_pos - 1]
    max_pos = subset.index(subset.max) + 1
    @list.move max_pos, last_pos
  end

  @list.items.should == original.sort
end
```

This new example requires us to be able to retrieve the current order of the jokes:

Download tubes/joke_list.rb
```
class JokeList
  def items
    num_items = @browser.get_xpath_count('//li').to_i
    (1..num_items).map {|i| @browser.get_text "//li[#{i}]/span[2]"}
  end
end
```

There's a lot more to interactive web pages than just drag and drop, of course. But we've touched on several places where Selenium does fairly well, including mouse input and waiting for state changes, both cornerstones of rich Internet apps.

12. Yes, it's $O(n^2)$ comparisons, but that's how people work.