Web Application Testing in Ruby

Introduction to Ruby – WATIR – Testing Web Applications

This talk will provide brief introductions to Ruby and the WATIR framework, and insights into their use in testing web applications.

Paul Carvalho

tester.paul at gmail [dot] com

KWSQA

Targeting Quality Conference 2009

Outline

What is Ruby?

- A full-featured object-oriented programming/scripting language
- Created by Yukihiro “Matz” Matsumoto in the mid-1990’s
- He wanted a scripting language “more powerful than Perl, and more object-oriented than Python.” And it had to be fun too.

Matz’ Philosophy

“Often people, especially computer engineers, focus on the machines. They think, “By doing this, the machine will run faster. By doing this, the machine will run more effectively. By doing this, the machine will something something something.” They are focusing on machines. But in fact we need to focus on humans, on how humans care about doing programming or operating the application of the machines. We are the masters. They are the slaves.”

Ruby Programs

- Ruby programs are interpreted rather than compiled
  - They are text files (with an .rb or .rbw extension)
- You can use any text editor. e.g.:
  - Notepad (in Windows)
  - Emacs (in Linux)
  - SciTE (included when you install Ruby in Windows)
- Use an IDE if you prefer. e.g.:
  - Eclipse
  - NetBeans
  - Komodo
  - Visual Studio

What can you do with Ruby?

- Ruby has many built-in libraries
  - Network access (http, ftp, ...), Email (SMTP, POP), XML
  - Database connectivity, Excel spreadsheets, CSV files
  - Graphics (Fox, TK), Math, ...
- The RAA (Ruby Application Archive) currently lists 1,700+ projects!
  - Standalone applications
  - Add-on Ruby libraries
- Ruby on Rails
  - Quick and easy-to-use framework for developing Web apps

The Ruby Language

- Everything in Ruby is an object
- Methods are like *verbs*... they are the *action* words that do stuff to the objects
  - Some examples:
    - book.read
    - dog.walk
    - taxes.pay
    - friend.sings ‘motown’
    - house.roof.leaky
    - name.capitalize
    - items.sort
    - container.empty?
    - todo_list.include? ‘party RSVP’
    - tree.leaves.sway

Interacting with Ruby

- Interactive Ruby Shell (irb)
    - Run from the command line
    - Enter Ruby commands to see what happens
    - Allows you to experiment with code in real time
    - With the Watir library, attach to browser windows and quickly identify page elements

Demo time!

How can you learn more?

- Lots of info available for free on the Internet:
  - Discussion forums, blogs and mailing lists
  - Free online tutorials and courses, videos
  - Cheat sheets, references, source code samples
- Books
- User groups
- Trial and error

- *See Reference Sheet handout*

What is WATIR?

- Web Application Testing in Ruby
  - (acronym is pronounced “water”)
- It is a free, open source Ruby library that drives a web browser the same way people do:
  - clicks links
  - fills in forms
  - presses buttons, and so on...
- It can be used to test all types of web applications (ASP, .Net, JSP, PHP, Rails, etc.)

What WATIR is not

- It is not a record/playback tool
  - Yes, you will be *programming* scripts
  - However, there are some recorders available:
    - Watir Extension Toolkit (WET)
    - FireWatir Recorder (TestGen4Ruby)
    - Webmetrics GlobalWatch Script Recorder
- Watir is not a test case management tool
  - You could probably create one in Ruby if you want
- No special software required to install on the servers
  - The scripts drive the web browser, wherever that might be

Why use WATIR?

- Free
- Simple, easy to learn and use, intuitive
  - You don’t need to be a professional developer to program Watir scripts
- Growing interest and development:
  - WatiN (.Net)
  - WatiJ (Java)
  - Celerity – run Watir tests without using a browser (fast)
  - Multiple browser support: IE, Firefox, Safari (Mac), Chrome
  - Flash application support (FlashWatir)

Why use WATIR (continued)

- Powerful. Watir uses Ruby, a full-featured programming language, not some proprietary vendor-script
  - You can connect to databases, read data files, export XML, structure your code into reusable libraries, and more
  - You can customise the scripts according to your needs
- Excellent Support (very active and helpful online forums)
- It's Free! ☺
- Read the Testimonials

Getting Started with WATIR

1. Download and Install Ruby
   - Installs on Windows, Linux, Mac
2. Install WATIR
   - Packaged as a gem
   - Gem = A Ruby library that can be installed over the internet
   - To install, type the following at a command prompt:

     ```
     > gem install watir
     ```
   - Register the AutoIt.dll (in Windows)
3. Download and install an HTML inspection tool. e.g.:
   - IE : IE Developer Toolbar, SpySmith
   - FF : Firebug

Now What?

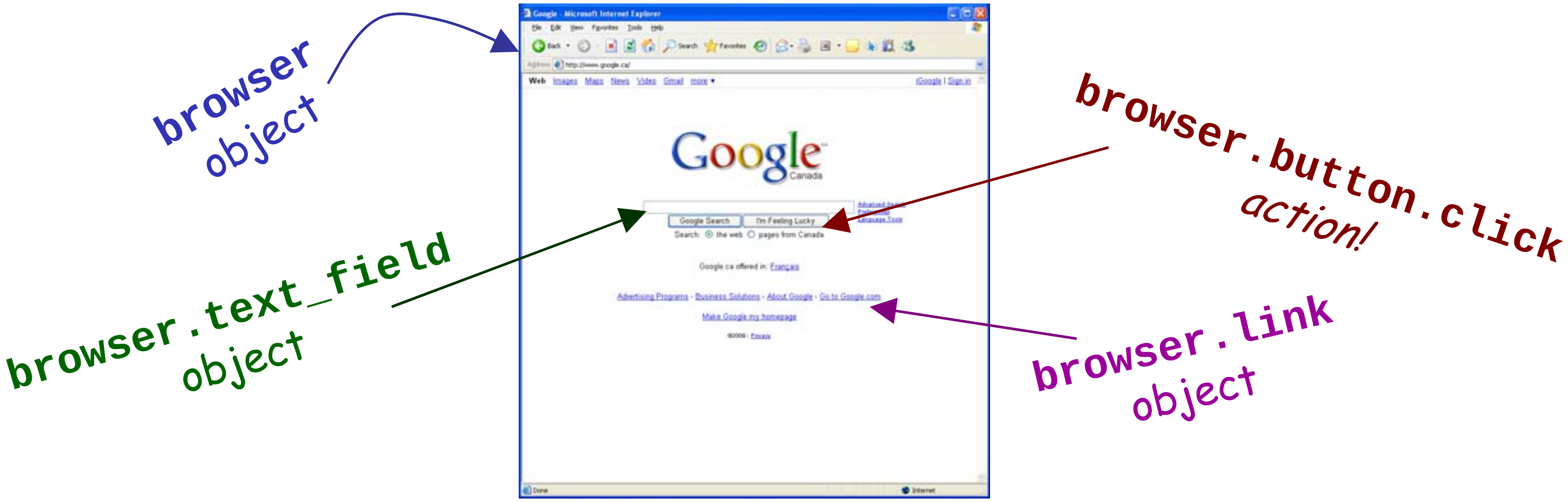
- Watir helps you automate the web browser
- So, what do you want to do?
  - Control the browser?
  - Find elements on the page?
  - Interact with elements on the page?
  - Scrape information off the page?

Controlling the Browser

```
# Always Load the Watir library at the top of your script
require 'watir'

# Start IE and navigate to a given URL
ie = Watir::IE.start( 'http://www.google.com' )

# or, attach to an existing IE window by title or url
ie = Watir::IE.attach( :title, 'Google' )
ie = Watir::IE.attach( :url, /regex matching url/ )

# Navigate to a different URL
ie.goto( "http://wtr.rubyforge.org" )

# You can also 'minimize', 'restore', and 'close' IE
ie.close
```

Identifying Page Elements

- Web pages contain objects:
  - links, buttons, tables, input fields, frames, etc.



- You can identify these objects in different ways:
  - View page source
  - Use a tool (e.g. IE Developer Toolbar)

How WATIR Finds Elements

- WATIR uses intuitive names for the common elements:

Text box	`ie.text_field( how, what )`
Button	`ie.button( how, what )`
Link	`ie.link( how, what )`
Drop-down list	`ie.select_list( how, what )`
Check box	`ie.checkbox( how, what )`
Radio button	`ie.radio( how, what )`
Table	`ie.table( how, what )`
Frame	`ie.frame( how, what )`

And many more... see Methods Supported by Element for a complete list

Finding the Exact Element

- *How*'s tell your method how to find the element you're looking for
- *What*'s tell your method the value for "how"

How's:	What's:
`:name`	• String value of "how"
`:id`	• /Regular expression/
`:index`	• variable
`:value`	
`:text`	
`:title`	
(and more)	

- So, instead of : **`dog.bark`**
- You can say : **`dog(:name,'Max').bark`**

Interacting with Elements

```
# Enter a few lines into a 'Notes' text field (or text area)
ie.text_field(:name,'Notes').set( "Watir \n Rocks!" )

# Select a specific value in a drop-down list box
ie.select_list(:id,/Size/).select( 'Medium' )

# Click the button with the specified value (a.k.a. label)
ie.button(:value,'OK').click

# Click the link matching 'text'
ie.link(:text,'Return to Home Page').click

# Access elements in a "frame" or "iframe"
ie.frame(:name,"Main").text_field(:name,'username').set('Alice')
```

Scraping Information (checking output)

```
# Get the title of the page, or the current URL
ie.title
ie.url

# Get the text content - for the whole page or just an element
ie.text
ie.span( how, what ).text

# Get all the HTML in the body of the page
ie.html

# Identify all the objects/elements on a page
ie.show_all_objects

# Return true if ‘text’ is on the page somewhere
ie.text.include?( 'text' )
```

Demo

Enough chatter, let's play!

(Check out /ruby/.../gems/watir../unittests/..)

Get More Information

- References:

WATIR :	http://wtr.rubyforge.org
Project Home :	http://wiki.openqa.org/display/WTR/Project+Home
Watir API Reference :	http://wtr.rubyforge.org/rdoc
Mailing list :	http://groups.google.com/group/watir-general

- *See Reference Sheet handout*

Organising Thoughts into Tests

- We know how to use Ruby to talk to web browsers
- Now we need to structure our scripts into tests
  - Need a *framework*
  - How do we *check* things? you know.. the actual *test* part?
  - How do we see the *results*?

Test Frameworks

- Ruby has a built-in framework: Test::Unit
- Others include:
  - Rspec - a Behaviour Driven Development framework for Ruby
  - Watircraft - Rspec & Cucumber and library structure
  - NUnit + Watir
- We won't be covering Domain Specific Languages (DSLs)

Ruby's Test::Unit Framework

- Test::Unit is a library of Ruby (just like Watir)
- It provides a way to organise your code into "tests"

- To use Test::Unit in your scripts you ‘require’ it just like you do with Watir:

```
require 'test/unit'
require 'watir'
```

- Test::Unit has built-in methods called “assertions” that help your tests with validation:

```
assert( browser.link(:text, 'Click Here').exists? )
```

Anatomy of a TestCase

- Simple example:

```
require 'test/unit'
require 'watir'

class GoogleHomePage < Test::Unit::TestCase
  def test_there_should_be_text_About_Google
    browser = Watir::IE.start "http://www.google.com"
    assert( browser.text.include?("About Google") )
  end
end
```

all Ruby

Tests are **methods** that start with "test_"

The Watir drops

"assertion" is a check point

Automatic Results Logging

- The previous example will produce results like the following when executed:

```
>ruby test_unit_example1.rb
Loaded suite test_unit_example1
Started
.
Finished in 2.709 seconds.
1 tests, 1 assertions, 0 failures, 0 errors
```

- Dot means the test passed
- 'F' means failure
- 'E' means error

Automatically records how long it took the script to complete

Automatic summary

Setup and Teardown

- The Test::Unit structure may include special methods that can be executed before and/or after every test

```
class SampleTest < Test::Unit::TestCase
  def setup
    # fill in code that will run before every test case here
  end
  def teardown
    # fill in code that will run after every test case here
  end
  def test_login
    # login test code, etc.
  end
end
```

Assertions == Expected Results

- Use ‘assert’ instead of writing your own **`if`** statement checks
- Test::Unit assertions include the following:
  - assert(boolean)   # fails if boolean is false or nil
  - assert_equal(1, 1)
  - assert_not_equal(1, 0)
  - assert_match(/test/, "test")
  - assert_no_match(/test/, "dfjlas")
  - flunk("force test to fail")   # forces test to fail
- Other libraries may extend the default set

Logging Results

- The default Test::Unit output is okay to start
- You can output test results in many different ways:
  - Text, CSV, Excel file (XLS), XML, HTML
  - Database
  - Graphs, charts
- You have the power to create whatever output format you need with Ruby

Weaknesses

- Test::Unit is fine for *unit* testing. However, it can be tricky for *system* testing where you have dependencies on the outcomes of previous tests.
- Default alphabetical test execution order is annoying
  - Watir library has new TestCase class so you can override this order
- Need to build calling scripts/framework if you only want to run a subset of tests
- Doesn’t provide guidance for organising large numbers of tests (e.g. 100’s+)

Test::Unit in Action

Demo time!

- For more information on Test::Unit, see Chapter 12 in the “Programming Ruby” book
- The chapter includes info on:
  - Assertions
  - Structuring Tests
  - Organizing and Running Tests

The End

Thank you for your time.

- For more information, see:
  - Ruby : http://www.ruby-lang.org/en/documentation
  - Watir : http://wiki.openqa.org/display/WTR/Project+Home
  - Google Groups: “comp.lang.ruby” and “watir-general”
- Or just write to me:
  - tester.paul at gmail [dot] com