Extracted from:

Cucumber Recipes

Automate Anything with BDD Tools and Techniques

This PDF file contains pages extracted from *Cucumber Recipes*, published by the Pragmatic Bookshelf. For more information or to purchase a paperback or PDF copy, please visit http://www.pragprog.com.

Note: This extract contains some colored text (particularly in code listing). This is available only in online versions of the books. The printed versions are black and white. Pagination might vary between the online and printer versions; the content is otherwise identical.

Copyright © 2012 The Pragmatic Programmers, LLC.

All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina

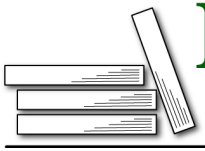
The Pragmatic Programmers

Cucumber Recipes

Automate Anything with BDD Tools and Techniques

Ian Dees,
Matt Wynne,
and Aslak Hellesøy

Edited by Jacquelyn Carter



Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and The Pragmatic Programmers, LLC was aware of a trademark claim, the designations have been printed in initial capital letters or in all capitals. The Pragmatic Starter Kit, The Pragmatic Programmer, Pragmatic Programming, Pragmatic Bookshelf, PragProg and the linking *g* device are trademarks of The Pragmatic Programmers, LLC.

Every precaution was taken in the preparation of this book. However, the publisher assumes no responsibility for errors or omissions, or for damages that may result from the use of information (including program listings) contained herein.

Our Pragmatic courses, workshops, and other products can help you and your team create better software and have more fun. For more information, as well as the latest Pragmatic titles, please visit us at *http://pragprog.com*.

Copyright © 2012 The Pragmatic Programmers, LLC.
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

Printed in the United States of America.
ISBN-13: 978-1-937785-01-7
Encoded using the finest acid-free high-entropy binary digits.
Book version: B1.0—August 15, 2012

Recipe 3

Run Slow Setup/Teardown Code With Global Hooks

Problem

You need to do something that takes a while before your first test, such as launching a browser or waiting for a desktop application to load. You're familiar with Cucumber's Before() hook, which runs once per scenario. But you want something that runs just once overall, so that your setup code doesn't slow down your test too much.

Ingredients

- Cucumber's built-in env.rb file for setup code
- Ruby's built-in at_exit() hook for teardown code[10]
- The Selenium WebDriver browser automation library[11]
- The Firefox web browser[12]

Solution

This recipe starts with a simple web testing project. Before we make our improvements, the code to start and stop the web browser executes inside regular Cucumber scenario hooks—and so the tests run more slowly than they should. We're going to see how to migrate that slow code to global hooks, so it only runs once.

You don't have to use any special hooks to run setup code when Cucumber starts. Just put your one-time startup code in env.rb, and Cucumber will run it before the first test.

That just leaves one question. With the Before() hook, there was a corresponding After() hook where you could shut down whatever application or browser you were using. Where do you put global teardown code that needs to run only once?

10. http://www.ruby-doc.org/core-1.9.2/Kernel.html#method-i-at_exit
11. http://seleniumhq.org/docs/03_webdriver.html#ruby
12. http://www.firefox.com

• Click HERE to purchase this book now. discuss

The answer is to use Ruby's built-in at_exit() method, which allows you to register a hook that runs just as Cucumber is exiting.

Let's look at a test that suffers from repeated setup code, and how you might convert it to use global hooks.

Setup

First, install Selenium WebDriver:

```
$ gem install selenium-webdriver
```

Now, create a simple test that has multiple scenarios:

global_hooks/bank.feature

```
Feature: Banking

  Scenario: Deposit
    Given I have $0 in my account
    # ...

  Scenario: Withdrawal
    Given I have $100 in my account
    # ...
```

Fill in a step definition that requires a web browser:

global_hooks/step_definitions/bank_steps.rb

```
Given /^I have \$(\d+) in my account$/ do |balance|
  @browser.navigate.to 'http://example.com/banking'
end
```

This code presumes that you've launched a browser and stored a reference to it in the @browser variable. The traditional approach to managing that variable is to use Before() and After() hooks. Let's look at that technique first, and then migrate to global hooks.

Scenario Hooks

Here's how you might have added per-scenario setup and teardown code without this recipe:

global_hooks/support/hooks.rb

```
require 'selenium-webdriver'

Before do
  @browser = Selenium::WebDriver.for :firefox
end

After do
  @browser.quit
```

```
end
```

Go ahead and run your feature, taking care to time the results. On Mac and Linux, you'd type the following:

```
$ time cucumber bank.feature
```

On Windows with PowerShell installed, you'd type this instead:[13]

```
C:\Hooks> Measure-Command {cucumber bank.feature}
```

You should see Firefox launch and exit before and after every step, and the total execution time will show it. It's time to migrate your startup code to global hooks.

Global Hooks

You're going to move your browser-launching code out of the Before() hook. But where to? You may recall that Cucumber is guaranteed to run code in env.rb before any of your other support code. That makes this file a good place for one-time setup.

The simplest approach is to run the setup code at file scope and store any state you need in a global:

global_hooks/support/env.rb
```
require 'selenium-webdriver'

$browser = Selenium::WebDriver.for :firefox
at_exit { $browser.quit }
```

Notice the symmetry between the creation of the $browser object and the registering of an at_exit() hook to tear it down when Ruby exits.

Before you run off and change your step definition to use the $browser global variable, it's worth considering the maintenance problems that globals can cause down the road. Take a moment to package this code up into a module, and change the global variable to a class-level attribute instead:

global_hooks/support/env.rb
```
require 'selenium-webdriver'

module HasBrowser
  @@browser = Selenium::WebDriver.for :firefox
  at_exit { @@browser.quit }
end
```

13. PowerShell comes with Windows 7 and can also be downloaded from http://www.microsoft.com/powershell.

Notice that you're now storing the browser in a class-level attribute @@browser, so that its value will be available across scenarios. In a minute, we'll add an accessor function for your step definitions to call.

First, though, take a look at the at_exit() hook. You're probably used to seeing these at file scope, so it may seem a little weird to use it inside a module definition. It will work just fine here.

Now, about that accessor function. Add the following code inside your module definition:

global_hooks/support/env.rb
```
def browser
  @@browser
end
```

One last thing: how do you make the browser() method available to your step definitions? By adding it to the *world*,[14] a container provided by Cucumber to store state between steps. You can do this by calling World() at file scope and passing it the name of your module:

global_hooks/support/env.rb
```
World(HasBrowser)
```

Don't forget to change your step definition to use the new browser() method:

global_hooks/step_definitions/bank_steps.rb
```
Given /^I have \$(\d+) in my account$/ do |balance|
  browser.navigate.to 'http://example.com/banking'
end
```

Now if you rerun your test, you should see that Firefox starts only once at the beginning of the run, and exits only once at the end. The total execution time will be cut almost in half.

Further Exploration

This recipe covered attaching hooks to the World object, which the Cucumber runtime creates for each scenario. For more on how you can customize this object's behavior, see Chapter 7 of *The Cucumber Book [WH11]*.

Most of the time, env.rb is the best place for global setup code. But if your hook must run specifically after configuration is complete, while still finishing before the first scenario runs, you can use the AfterConfiguration() hook instead.[15]

14. https://github.com/cucumber/cucumber/wiki/A-Whole-New-World
15. https://github.com/cucumber/cucumber/wiki/Hooks