

# Functional testing for Web applications

## Using Selenium, Windmill, and twill to test Google App Engine applications

Skill Level: Intermediate

Noah Gift ([noah.gift@giftcs.com](mailto:noah.gift@giftcs.com))

Founder GiftCS, LLC

GiftCS

10 Mar 2009

Master automated, functional testing using the open source tools Selenium, Windmill, and twill. Using the techniques in this article, learn how to perform functional testing on the Google App Engine, blogging software, or your own home-grown application.

### Introduction

#### Notes on twill and Google App Engine

From [Resources](#), you can view a more detailed article on getting a "Hello World" application running from scratch. On the OS X platform you can download the Google App Engine Launcher, which will create the boilerplate files for a "Hello World" application for you.

To get started using twill, you only need to run the command: `sudo easy_install twill`. Next, simply invoke the `twill-sh` scripting tool to interactively test your application.

Most developers, whether they test their code or not, have at least gotten a lecture about testing code at some point. Web developers—more so than most developers—need to deliver applications quickly, so unit testing often takes a backseat to a deadline. In some circles, it is never all right to skip unit testing any code, as a unit test tests actual components of the application and provides a way of explaining the internal workings of the code to other developers. Functionally testing your Web application is quite a different story though, and for some reason hasn't

gotten as much of a buzz.

In this article, I explore several different tools that help you perform functional testing on your Web application. I use Google App Engine here, but the testing techniques will apply to any Web application. I will also argue that it is never all right to forgo functional testing, because it is so quick and so easy to perform, at least a minimal level of functional testing. In this article, I explore three functional testing tools: Windmill, Selenium, and twill. Both Windmill, and Selenium are functional Web testing frameworks that allow automation of user interface testing in a Web browser for JavaScript and Asynchronous JavaScript and XML (Ajax) applications. Twill is a lightweight Web scripting tool that deals with non-JavaScript functional testing.

## Functional testing with twill

### Easy Install

Easy Install is a tool that helps install Python packages, either locally or on the network. Most often, the `easy_install` command is used to install packages from the network.

I'll start the discussion of functional testing with the lightweight command-line Web browser and scripting tool, twill, and a default Google App Engine project.

The first thing you do is establish a connection to your application. To do that, use the `go` command, as shown in Listing 1. Note, if you enter `show`, it then shows the actual output.

### Listing 1. Sample show output

```
# twill-sh
== Welcome to twill! ==
>> go localhost:8087
==> at http://localhost:8087
current page: http://localhost:8087
>> show
Hello World!
current page: http://localhost:8087
```

Another interesting feature of twill is its ability to check `http` status codes.

### Listing 2. Sample twill output of http status codes.

```
> go http://localhost:8087
==> at http://localhost:8087
current page: http://localhost:8087
>> code 200
current page: http://localhost:8087
>> code 400
ERROR: code is 200 != 400
current page: http://localhost:8087
```

As you can see from the output of the command, it only returns an error if it gets a status code that doesn't match what is expected. Twill also supports the ability to run

these actions as a script. You can name the file anything you want and pass it to `twill-sh`. If you place those commands in a file called `test_twill.script`, you will see something like Listing 3.

### Listing 3. More sample twill output

```
# twill-sh test_twill.script
>> EXECUTING FILE test_twill.script
AT LINE: test_twill.script:0
==> at http://localhost:8087
AT LINE: test_twill.script:1
--
1 of 1 files SUCCEEDED.
```

Twill is a handy tool for automated testing of the non-JavaScript portions of your Web application, and it has more features than what I covered, such as the ability to work with variables, cookies, forms, http authentication, and more. If you would like to know about more advanced use cases, see [Resources](#).

### Functional testing with Selenium

Selenium is a heavier-weight testing tool that allows cross-platform testing in the browser. Writing cross-platform JavaScript code is a regrettable cross that Web developers must bear. Writing Web applications is difficult enough, and just when you think you're done, you inevitably run into some obscure bug that is only present on one browser.

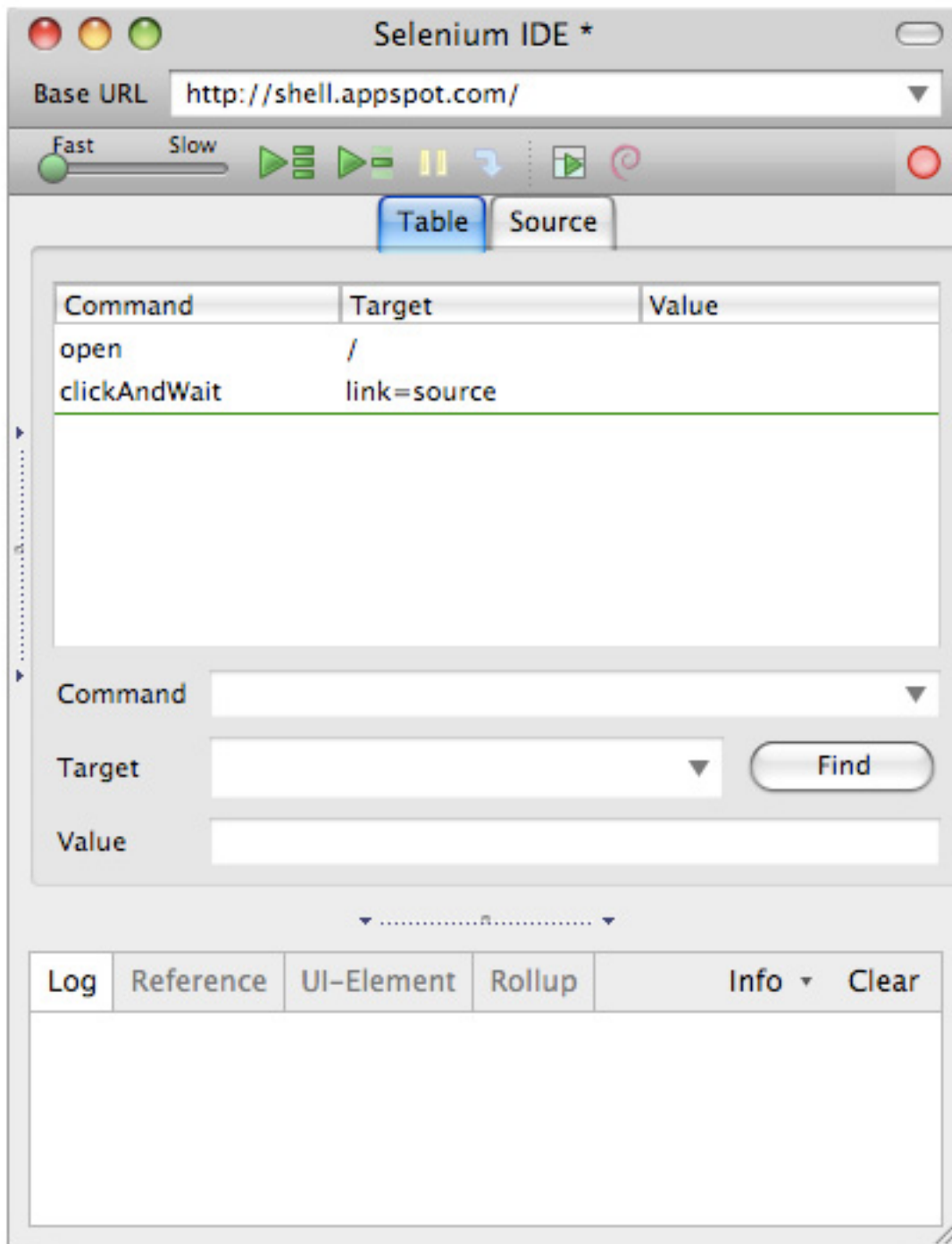
Unfortunately, a unit test does not catch this type of bug. In fact, these bugs often make a Web developer skeptical of doing any testing in the first place. They figure that testing is yet another hoop to jump through, that it gets in the way of their already tight deadline, and it doesn't even reliably work. So why bother?

Selenium (as well as other browser testing tools) is an answer to this problem. You can write functional tests that run in each browser, and then implement some form of continuous integration system that runs the functional tests upon each check-in of the source code. This allows potential bugs in the browser to quickly get caught, and has an immediate payoff.

One of the most basic things Selenium does is record browser actions so they can be replayed as a test. Looking at the sample in Figure 1, you see a window where you place a base url. From there you simply record the actions of the Web site you are testing. In this case, I am testing a Google App Engine site:

`http://shell.appspot.com`, which is a demo Ajax Python interpreter. After the session has been recorded, you can then export the tests in Python and run them using Selenium.

### Figure 1. Selenium IDE window



To run the tests, simply save your tests as Python code (or whatever language you choose), download and run the Selenium RC test server, and then run your tests. Listing 4 shows an example of what that test might look like.

**Listing 4. Example Selenium test**

```
from selenium import selenium
import unittest, time, re

class NewTest(unittest.TestCase):
    def setUp(self):
        self.verficationErrors = []
        self.selenium = selenium("localhost",
            4444, "*chrome", "http://shell.appspot.com")
        self.selenium.start()

    def test_new(self):
        sel = self.selenium
        sel.open("/")
        sel.click("link=source")
        sel.wait_for_page_to_load("30000")

    def tearDown(self):
        self.selenium.stop()
        self.assertEqual([], self.verficationErrors)

if __name__ == "__main__":
    unittest.main()
```

You can launch Selenium RC, which acts as a proxy for testing, on multiple browsers, and then run your functional test. Listing 5 shows what the output of Selenium RC looks like.

### Listing 5. Sample of Selenium RC output

```
# java -jar selenium-server.jar
01:18:47.909 INFO - Java: Apple Inc. 1.5.0_16-133
01:18:47.910 INFO - OS: Mac OS X 10.5.6 i386
01:18:47.915 INFO - v1.0-beta-1 [2201], [1994]
01:18:48.044 INFO - Version Jetty/5.1.x
01:18:48.045 INFO - Started HttpContext[/,/]
01:18:48.047 INFO - Started HttpContext[/selenium-server]
01:18:48.047 INFO - Started HttpContext[/selenium-server/driver]
01:18:48.055 INFO - Started SocketListener on 0.0.0.0:4444
[output suppressed for space]
```

I encourage you to read the full Selenium RC FAQ to understand how it interacts with multiple browsers (see [Resources](#)). As you can see, using Selenium to automate cross-platform functional tests is quite easy, and there is support for many languages including plain HTML, Java™ code, C#, Perl, PHP, Python, and Ruby.

### Windmill

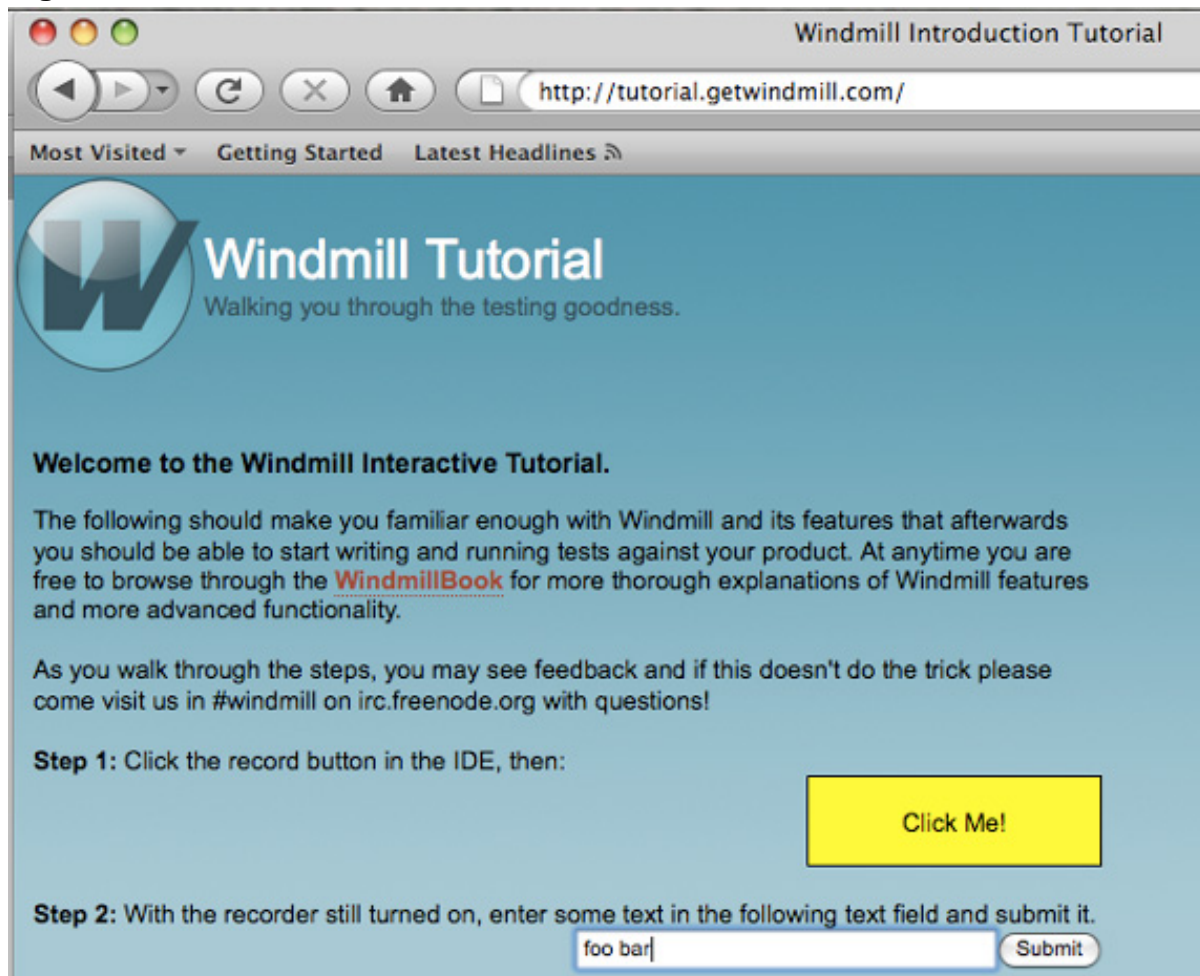
Windmill is a Web testing framework that is very similar to Selenium, although there are some differences. One main difference is that Windmill is written in Python and JavaScript, and is developed within the umbrella of the Chandler project, an ambitious open source competitor to Microsoft® Outlook.

To get started using Windmill, simply run the command: `sudo easy_install windmill`. This installs the windmill testing framework. Next, if you enter `windmill`

firefox, the Windmill IDE opens (see Figure 2). Then the testing page opens, as shown in Figure 3. In Figure 2 you can see that the IDE records actions much like Selenium. You can then save the test file, which looks like the output in Listing 6.

**Figure 2. Windmill IDE window**



**Figure 3. Windmill tutorial****Listing 6. Sample test file output**

```
# Generated by the windmill services transformer
from windmill.authoring import WindmillTestClient
def test_recordingSuite0():
    client = WindmillTestClient(__name__)
    client.click(id=u'recordedClickId')
    client.click(id=u'textFieldOne')
    client.type(text=u'foo bar', id=u'textFieldOne')
    client.click(id=u'btnSub')
```

There is an option to save a test as JSON or Python, and in this case I save it as Python. Next, to actually run the test, you simply run the test file from the command line with the test option. Listing 7 shows how this looks.

**Listing 7. Running the test**

```
# windmill firefox test=windmill_test.py
http://tutorial.getwindmill.com/
Started ['/Applications/Firefox.app/Contents/MacOS/firefox-bin',
```

```
'-profile', '/var/folders/1K/  
1KgyCzqJHButzT6vq8vwHU+++TI/-Tmp-/tmp_ovtnN.mozrunner',  
'http://tutorial.getwindmill.com/windmill-serv/start.html']  
Server running...
```

You can easily adapt this example to your Google App Engine application or any other Web application, for that matter. Windmill also has great documentation that explains how to do more advanced tasks such as extending windmill with plug-ins.

## Conclusion

Functional testing is essential to the process of Web development. Without functional testing, Web development becomes a guessing game that can be filled with frantic, error-prone deployments and refactoring.

So, should functional testing be mandatory for all Web developers? I would argue, yes, all Web applications should be tested. Lack of functional testing for Web applications is a definite red flag considering how easy it is to do—at least a minimum of testing—with Selenium, Windmill, or twill. To borrow a tag line from the author of twill, Dr. Titus Brown, "if you don't test your code how you can say it works?"

## Downloads

Description	Name	Size	Download method
Sample Code For This Article	functional_testing_7051e.zip	7051e	<a href="#">HTTP</a>

[Information about download methods](#)

## Resources

- [Python For Unix and Linux System Administration](#): Use this book to develop your own set of command-line utilities with Python to tackle a wide range of problems.
- "[Getting Started With Google App Engine](#):" This article gives you a good introduction to the Google App Engine.
- [Google App Engine in Action](#): Use this book to create applications with a custom feel to them using Python.
- [Google App Engine](#): Read the official documentation.
- [Hello World Google App Engine](#): Read a more detailed article on getting a "Hello World" application running from scratch.
- [Python Tutorial](#): Take this tutorial to learn the basic concepts and features of the Python language and system.
- [Selenium RC Server](#): Get more information on the Selenium RC Server.
- [twill](#): Get more information on the twill scripting language.
- [Windmill- Test Automation](#) : Find out more about the Windmill testing framework.
- [An Introduction to Testing Web Applications With twill and Selenium](#): Use this resource to quickly start writing your own functional tests.

## About the author

Noah Gift

Noah Gift is the co-author of "[Python For Unix and Linux System Administration](#)" by O'Reilly, and is also working on "[Google App Engine In Action](#)" for Manning. He is an author, speaker, consultant, and community leader, writing for publications such as IBM developerWorks, [Red Hat Magazine](#), [O'Reilly](#), and [MacTech](#). His consulting company's Web site is <http://www.giftcs.com>, and much of his writing can be found at <http://noahgift.com>. You can follow him at twitter at <http://twitter.com/noahgift>

He has a Master's degree in CIS from Cal State in Los Angeles, California, a Bachelor of Science degree in Nutritional Science from Cal Poly in San Luis Obispo, California, is an Apple and LPI certified SysAdmin, and has worked at companies such as, Caltech, Disney Feature Animation, Sony Imageworks, and Turner Studios. He is currently working at [Weta Digital](#) in New Zealand. In his free time he enjoys spending time with his wife Leah, and their son Liam, composing for the piano, running marathons, and exercising religiously.

## Trademarks

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.