LLNL-CONF-578774

Web-based Testing for an Environmental Information Management System

E. Barbosa
Inter American University of Puerto Rico

G. W. Laguna
Lawrence Livermore National Laboratory

August 31, 2012

Society for Advancement of Chicanos and Native Americans in Science (SACNAS) National Conference
Seattle, WA, United States
October 11, 2012 through October 14, 2012

Lawrence Livermore National Laboratory is operated by Lawrence Livermore National Security, LLC, for the U.S. Department of Energy, National Nuclear Security Administration under Contract DE-AC52-07NA27344.

Disclaimer

Web-based Testing for an Environmental Information Management System

Selenium and Python

Elizabeth Barbosa[1]
Inter American University of Puerto Rico, Bayamon, PR, 00957

Gary Laguna[2]
Environmental Restoration Department, Lawrence Livermore National Laboratory, Livermore, CA, 97550

[1] Computer Scientist, Environmental Restoration Department, LLNL, Inter American University of Puerto Rico
[2] Computer Scientist, *mentor* Environmental Restoration Department, Lawrence Livermore National Laboratory, CA
August 6, 2012

Abstract

As the World Wide Web has evolved, so have web-based applications. Early web applications were relatively simple with limited variability and interactivity, but present day web applications are typically comprised of hundreds to thousands of dynamic and variable software components. The magnitude and scope of today's web-based applications require testing tools that can operate in the web realm that are flexible, repeatable, and can be automated. Testing is an essential step in the development process necessary to validate software. The principle project for my 2012 summer internship at Lawrence Livermore National Laboratory (LLNL) was to develop tests for TEIMS using Selenium. Selenium is an open source web-based integrated testing tool and framework based on the associated language Selenese. TEIMS is an enterprise system comprised of a collection of web-based applications and back-end database that support data collection, reporting and, scientific findings for ERD.

I used Selenium to develop dozens of tests for TEIMS web applications. After completing the development of a suite of tests for an application, I exported each suite to the Python language. I then developed a shell script that allows all test suites to be run with a single command. The initial version of this rollup script would run each test independently, with the result that each test required user authentication to the web server. Subsequent versions of the rollup script overcome this drawback by creating a single web-browser resource that is used to run each test. This resulted in a script that only required a single user authentication to the web server, allowing all remaining tests to be run without additional user interaction. The final outcome of this work turned testing for the TEIMS project from a manual, time-consuming task to an easier to manage automated process. The result is powerful regression testing with a simple terminal command.

Nomenclature

IDE = *Integrated Developing Environment*

TEIMS = *Taurus Environmental Information Management System*

ERD = *Environmental Restoration Department*

I. Introduction

World War II era operations (1940s) at the U.S. Navy NAS Livermore, contributed to environmental contamination at the LLNL site. LLNL was established in 1952 as a national defense research and development laboratory. Since the discovery of contamination in 1983, LLNL has actively pursued environmental restoration. The TEIMS system has evolved to facilitate these efforts, starting with predecessor applications developed in the 1980's. TEIMS is a collection of web-based applications and tools that have been developed through subsequent years. Software testing for web-based systems such as TEIMS has lagged behind the explosive growth in size and complexity of the applications. Selenium is an open source testing solution that is gaining in capability and popularity for web-based testing, which makes Selenium a great choice to meet TEIMS testing needs.

II. Testing with Selenium IDE

Software testing is a process conducted to provide stakeholders the assurance of software functionality. Additionally, effective testing techniques are necessary to enhance, maintain, and validate software. Historically, developers tested TEIMS applications and modules manually, which was a time-consuming process providing limited coverage. This manual testing process had to be repeated for each software release.

As a first step toward automated testing, the Selenium IDE was used to create multiple tests for selected TIEMS applications. The Selenium IDE is a complete integrated environment for developing selenium tests. Implemented as a Firefox extension, the Selenium IDE facilitates the recording, editing and debugging of tests. The IDE records tests in Selenese, a scripting language specific to Selenium. The Selenium IDE is installed as an add-on to the Firefox browser downloadable from, seleniumhq.org.

Selenium Features:

- Record and playback
- Intelligent field selection

- Auto complete for all common Selenium commands
- Walk through tests
- Debug and set breakpoints
- Save tests as Ruby, Python, C#, Java
- Option to automatically assert the title of every page

III. Creating a Test

Creating tests using the Selenium IDE is a straightforward process. During recording, the Selenium-IDE will automatically insert commands into your test case based on your actions, see Figure 1.

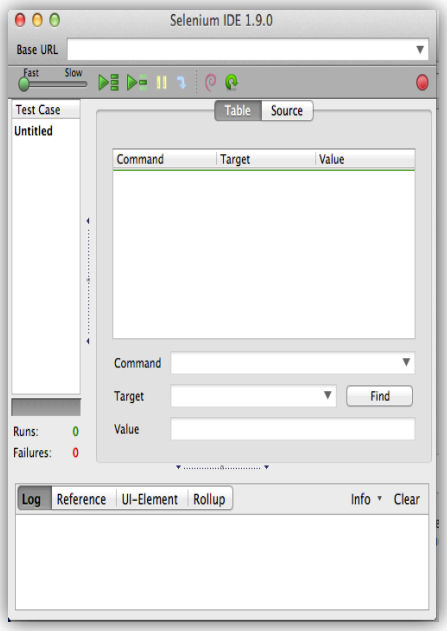
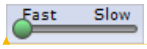


Figure 1. Selenium IDE testing framework environment.



Test Speed Control Slider: Adjusting this slider allows control of how fast Selenium will cycle through commands.

Run All Tests: Clicking on this button will run all tests in the current test suite, left test case pane.

Run One Test: Clicking on this icon will run the currently opened or selected test case.

Pause/Resume: This button allows a running test to be stopped and subsequently resumed.

Run One Command: This button allows the test to be "stepped" through one command at a time.

Record Button: This button turns the record mode on or off. When recording, Selenium will record the user's browser actions.

Unknown
Formatted: Font:11 pt

Developing a new testing framework for TEIMS required more than just recording. The functionality of the recorded tests needed to be extended by using Selenium commands. These commands are classified into three types: *Actions*, *Accessors,* and *Assertions*.

- *Action* commands allow the web application to be manipulated. These commands are used to perform actions such as *click* and *select* elements on the web page. *Action* commands that fail, or have errors will abort the execution of the test case.
- *Accessor* commands are generally used to store process state in variables, which can then be used with Assertions.
- *Assertions* are Selenese commands that verify test applications are doing what they are expected.

While developing test scripts, *test cases* are displayed in the test case pane, see Figure 2. This has two tabs, one for displaying the commands in Selenese, while the other displays the tests in HTML. Test commands can also be exported to a supported programming language by the framework.

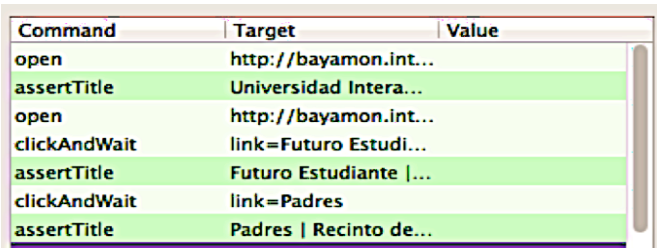


Figure 2, Test Case Pane

When running test cases, error messages and information messages showing progress are displayed in the log pane automatically. These messages are useful for test case debugging, see Figure 3.

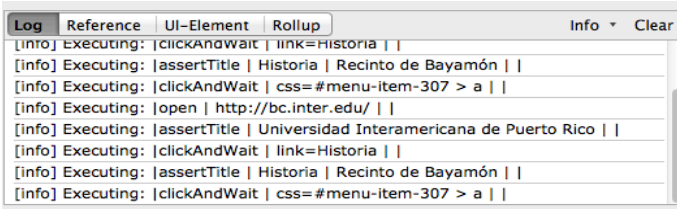


Figure 3, Log Pane.

The Reference tab is the default selection when creating or modifying Selenese commands and parameters in Table mode. While in Table mode, the Reference pane will display documentation on the current command, see Figure 4.

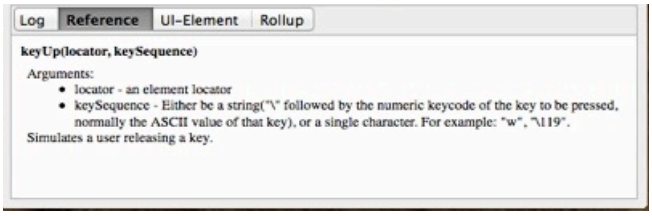


Figure 4, Reference Pane.

IV. Debugging Selenium Tests

As with any software development activity, Selenium tests will occasionally need to be debugged. Below are some guidelines for debugging Selenium scripts.

- General errors – errors can sometimes be fixed by adjusting the fast/slow bar at the left top of the IDE. Some tests cannot be run fast due to the time required to retrieve and load data. If a test is not working, try placing the speed marker underneath the (st) of “fast”.
- Toggle a Breakpoint in a test – this can be helpful when working with very long tests. Setting breakpoints allows execution to be stopped somewhere inside the test script. Once stopped, individual tests can be stepped through one at a time to help isolate and identify errors. A breakpoint can be set by going to the desired line in the script and pressing (b) on the keyboard.
- Clear Start Point - pressing (s) in your keyboard sets the start point for the test script. This can be useful with breakpoints, allowing the user to work on a particular section of the script without having to start over from the beginning and its startup overhead such as authentication and data loading.
- Run One Command – While debugging it is useful to use this button, this will allow you to “step” through one command at a time.

- Execute - pressing (x) on a highlighted line of code causes that line to be executed. Alternatively, double clicking on a line of code or “pressing the run one command” button will have the same effect.
- Timing “errors”: a common error is caused by not giving the click and wait command enough time to receive and load the data your application needs. The default timeout value can be modified by going to the Options menu and changing the timeout value. Timing can also be set for a particular script by adjusting the appropriate Selenese commands.
- Javascript - Some web applications have Javascript attached to input elements that execute upon data entry. The keyUp command must be used to force this event. To use the Selenium KeyUp command, the name of the input box should be used as the target, and use “10” for the value. This sends the same ASCII command as if user had pressed the “enter” key.
- Back function – the Selenium IDE does not record the back function in the browser. One way to simulate this behavior is by typing the “goBack: command in directly in Selenium. This will simulate the user clicking the "back" button in their browser.

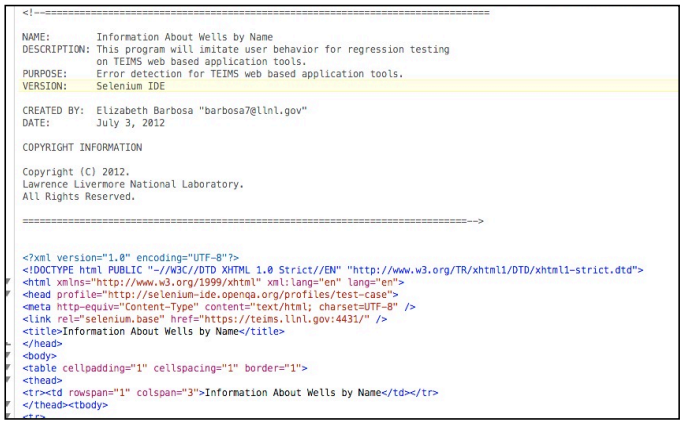


Figure 5, This is a test created in Selenium. Viewed as an HTML file.

V. Exporting Tests to Python

Once the test script is completed, the tests are exported to the Python language (Figure 6). To execute the tests in Python requires some set up on the computer to be used for testing. In particular, the Python Client Web Driver must be downloaded, installed and configured.

```
GAVT.py  ⋮  test_g_a_v_t  ⋮
#============================================================================

# NAME:         GAVT test
# DESCRIPTION:  This program will imitate user behavior for regression testing
#               on TEIMS web based application tools.
# PURPOSE:      Error detection for TEIMS web based application tools.

# CREATED BY:   Elizabeth Barbosa "barbosa7@llnl.gov"
# DATE:         August 3, 2012

# COPYRIGHT INFORMATION

# Copyright (C) 2012.
# Lawrence Livermore National Laboratory.
# All Rights Reserved.

#============================================================================

from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import Select
from selenium.common.exceptions import NoSuchElementException
import unittest, time, re

class GAVT(unittest.TestCase):
    def setUp(self):
        self.driver = webdriver.Firefox()
        self.driver.implicitly_wait(30)
        self.base_url = "https://teims.llnl.gov:4431/"
        self.verificationErrors = []

    def test_g_a_v_t(self):
        driver = self.driver
        driver.get("https://teimsdev.llnl.gov:4431/cgi/data_mining/home.pl")
        self.assertEqual("Data Mining Tools", driver.title)
        driver.find_element_by_xpath("(//a[contains(text(),'LLNL')])[3]").click()
        self.assertEqual("GIS Analysis and Visualization Tool", driver.title)
        # Selection of each tool option on LLNL
        # Verification of left panel options
        driver.find_element_by_id("releaseBlurb").click()
        # ERROR: Caught exception [ERROR: Unsupported command [selectWindow]]
        self.assertTrue(self.is_element_present(By.XPATH, "//div[@id='contentFade']/ul/li[2]"))
        driver.find_element_by_id("bgFade").click()
        driver.find_element_by_id("gavtDoc").click()
```

Figure 6. Exported test into Python.

VI. Executing Selenium Tests

The Selenium Web Driver allows Selenium to be used with other languages. The Selenium Python Client Driver is a Python language binding for Selenium (version 1.0 and 2.0). All Selenium downloads can be found at seleniumhq.org.

- version 1.0 = Selenium RC
- version 2.0 = Selenium Web driver

1. Download the last version of Selenium Web driver (currently Python 2.25.0).
2. Extract the content of the downloaded zip file.
3. Copy the module with Selenium's driver for Python (selenium.py) to the folder *C:/Python25/Lib* (this will allow import directly to any script).
4. The module will be in the extracted folder - *selenium-python-driver-client*.

Installing the Selenium Web Driver provides the flexibility to run each test by just executing a script. The web driver instantiates a browser container for running each test. The execution of tests one at a time can still be a slow process when hundreds of web pages need to be tested.

VII. Regression Testing

The next challenge was to be able to run all tests with a single command. Developing a solution to this problem would provide an automated process for regression testing that is flexible and durable. Regression testing is the process of re-testing existing software's functional and non-functional capabilities after changes, such as enhancements, patches or configuration changes.

PyUnit testing frameworks and modules are used in the process of wrapping these tests into a modern, easy to use Test Suite. PyUnit is a framework, based on JUnit that makes it easy to write automated Test Suites in Python. All of these components have been used to develop a powerful Python script that will look for all tests and execute them each in turn (Figures 7, 8 & 9). Python scripts are pieces of code that can be written and run without having to be compiled into the software.

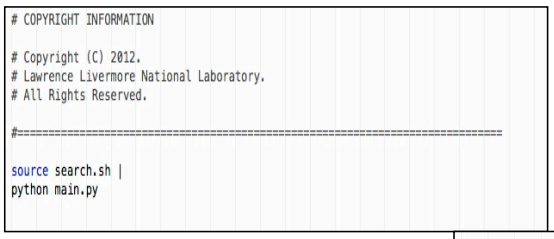
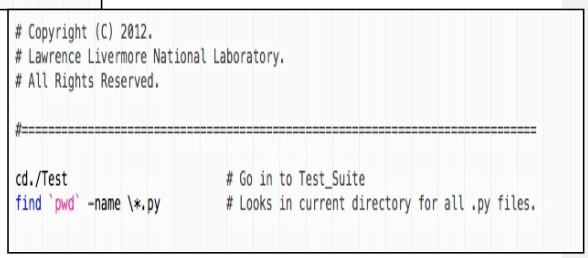


Figure 7. First step of the shell script. This script will run a search script, main program and return.

Figure 8. Second step, shell script. Looks through directory for all test cases.



```
# COPYRIGHT INFORMATION

# Copyright (C) 2012.
# Lawrence Livermore National Laboratory.
# All Rights Reserved.

#=============================================================================
import sys, os



testSuite = sys.stdin.readlines()       # Call for module and store return in testSuite.
test = list()                           # Assign list to test.


for items in testSuite:                 # Iterates through testSuite
    size = len(items)-1
    test.append(items)                  # Push items in test.

for x in test:                          # Iterates through test and executes each.
    os.system('python ' + x)
```

Figure 9. This program will collect all test cases and execute them.

VIII. Conclusion

Regression testing for TEIMS is now flexible, consistent, and easy to run, with continually improving coverage. A terminal-based log records failure or success for each test and additional details for failure. The result is powerful regression testing with a single terminal command.

Acknowledgments

I would like to thank my mentor Gary Laguna and also my technical mentors; Julia Britt, Francesca Demello, Ahn Tu Quach and Suzie Chamberlain, for the support and guidance you have provided me. Thanks to the Institute for Scientific Computing, ISCR for supporting my visit and especially the organization, Lawrence Livermore National Laboratory. I would also thank my funding organization, the National Nuclear Security Administration, NNSA Consortium, (SHPE) for providing the funds needed to make this knowledge enriching experience possible.

References

Selenium IDE Retrieved June 4, 2012 from http://seleniumhq.org/

Selenium 2.0 Documentation (2011) Retrieved June 15, 2012 from Selenium Reference retrieved June 18, 2012 from http://release.seleniumhq.org/selenium-core/0.8.0/reference.html

http://selenium.googlecode.com.svn/trunk/docs/api/py/index.html#python-client

Interface Selenium Retrieved June 20, 2012 from http://release.seleniumhq.org/selenium-remote-control/0.9.2/doc/java/com/thoughtworks/selenium/Selenium.html

Git Hub Developer, *Gits API* Retrieved July 10, 2012 from http://seleniumpython.readthedocs.org/en/latest/api.html

Python Beginners Guide May 7, 2012 Retrieved July 30, 2012 from http://wiki.python.org/moin/BeginnersGuide

Selenium Python Bindings Documentation Retrieved July 28, 2102 from http://selenium-python.readthedocs.org/en/latest/api.html

Python Bindings, *A guide to using the python bindings for Selenium/Webdriver.* (May 21,2012) Retrieved August 2, 2012 from http://code.google.com/p/selenium/wiki/PythonBindings

PyPI - the Python Package Index Retrieved August 2, 2012 from http://pypi.python.org/pypi

Python 2.7.3 documentation, *25.3 Unittest- Unittest testing Framwork,* August 8, 2012 Retrieved from http://docs.python.org/library/unittest.html

Steve Purcell (August 10, 2001) PyUnit - the standard unit testing framework for Python retrieved July 31, 2012 from http://pyunit.sourceforge.net/