Editor: Christof Ebert ■ Vector Consulting ■ christof.ebert@vector-consulting.de

Web Application Tests with Selenium

Andreas Bruns, Andreas Kornstädt, and Dennis Wichmann

Web applications tend to continuously evolve and thus need thorough, yet lean and automatic, regression testing. In this installment of Software Technology, Andreas Kornstädt and his colleagues describe automatic regression testing for Web applications that uses the Selenium testing framework. Selenium is portable open source software available for Windows, Linux, and Macintosh. Tests are written as HTML tables or in a number of programming languages and can run directly in most Web browsers. Andreas and his colleagues also provide many useful testing hints for practitioners. I look forward to hearing from both readers and prospective column authors about this column and the technologies you want to know more about. —Christof Ebert

We've all been there: We've written a technically perfect application, the green bar shows us all unit tests have passed, and the coverage tool reports the degree of coverage we aimed at. Nevertheless, the (intended) users aren't really impressed and tell us that the application might be a nice piece of software but it's not what they expected. Situations like this brought acceptance testing to the fore.

However, manual acceptance testing can be tedious. To provide automated acceptance testing of Web applications, particularly those using Ajax (Asynchronous JavaScript and XML), the set of open source Selenium tools comes in handy (see Table 1). With these tools, developers can easily run acceptance tests in their Web browsers.

Acceptance Testing

Acceptance tests completely leave out a program's inner machinations and focus solely on the effects that affect the user. They describe exactly what the system should and shouldn't do to gain user acceptance.

Depending on the system, this "what" can be anything from initiating specific jobs in a printing factory to sending out text messages to cell phones. However, in most cases user acceptance hinges on what can be seen on and done with the user interface. For example, users will expect that an email program's "send" button will be active only when at least one recipient has been specified and that copies of sent messages will be visible in the "sent" folder.

Unlike unit tests, acceptance tests take the form of a step-by-step script that acceptance testers walk through while sitting in front of the application under test. As with any test, there can be some setup (a database with some dummy users) and a teardown.

Although many companies never really go beyond this stage, tests show their true power (and soothing qualities) only when run as automatic regression tests. In a Java environment, GUI test tools include Jemmy, SWTBot, and Abbot, and tools for testing Web applications include HtmlUnit or HttpUnit (For URLs for these and other tools, see the "Resources" sidebar). In general, these tools come in two flavors: *capture-and-replay* versus *programmatic*. Capture-and-

 0740-7459/09/$26.00 © 2009 IEEE

replay tools are great for ensuring that a certain scenario leads to the same results that it did previously. They do this by recording a real user's actions as he or she walks through the script and replaying those actions in the application. You can effortlessly specify tests that way, but the nightmare begins if the scenario or the software changes and the recording must be adapted. Programmatic tests take more time to craft, but making changes doesn't cause major hassles. These tests are more flexible and can be written in a test-first manner, without the application under test being actually there.

Testing Web Applications

The classic way to test Web applications was on the protocol level with HttpUnit and HtmlUnit. The tester created an HTTP request, sent it to the Web site under test, and analyzed the response (usually in HTML). The advent of Rich Internet applications (RIAs) using Ajax made handling these classic tools much more cumbersome. Instead of a single request-response cycle, test authors must coordinate an arbitrary number of those cycles in parallel.

Selenium Core

Following the shift from the protocol level to the user-interaction level, testing what can actually be seen and done in the browser has more appeal. Selenium Core achieves this by running a JavaScript application in a host browser and controlling the Web application under test using that browser's capabilities. It's available for the current and previous versions of Internet Explorer, Firefox, Safari, and Opera.

The most basic way of interacting with Selenium Core is sending it commands in Selenese, Selenium's own control language. The commands are grouped together in an HTML table with columns for the command (`type`, `clickAndWait`), the target element (`amount`), and optionally a value. Table 2 lists some useful Selenese commands.

A Remote Control for Browsers

Producing HTML tables and learning Selenese vocabulary isn't to everyone's taste, so Selenium provides Remote Control (RC) applications in C#, Java, Perl, PHP, Python, and Ruby. Each Selenium RC API offers matching calls for all Selenese commands plus commands to start, stop, and configure browsers.

Figure 1 shows how to test autocompletion of country codes using Selenium RC for Java. Besides methods for controlling the browser (`start`, `open`, `stop`), checking the browser's state (`getTitle`, `isElementPresent`), and manipulating the application (`click`, `type`), Selenium RC offers utilities for common tasks such as waiting (through built-in utilities such as `waitForPageToLoad` or by customizing commands with `Wait`).

Table 1

Selenium tools

Tool	Purpose
Selenium Core	Modify and check an Ajax application using commands in Selenese, Selenium's control language.
Selenium RC	Remotely control Selenium Core using a common programming language.
Selenium Grid	Use several remote controls in parallel to expedite testing.
Selenium IDE	Capture and replay tests from within Firefox.

Resources

The most complete source of information about Selenium is at http://seleniumhq.org/docs.

Java Power Tools, by John Ferguson Smart (O'Reilly 2008), contains a substantial chapter on Selenium. It also shows how to easily integrate Selenium with JUnit, Ant, and Maven. But be warned: the book has chapters on many other tools and totals over 900 pages. On the other hand, *An Introduction to Testing Web Applications with twill and Selenium*, by C. Titus Brown, Gheorghe Gheorghiu, and Jason Huggins (2007), from the O'Reilly Short Cuts series, is concise but, owing to its nature, can't provide in-depth coverage.

Wikipedia briefly but adequately describes acceptance testing (http://en.wikipedia.org/wiki/Acceptance_testing), but *Fit for Developing Software: Framework for Integrated Tests*, by Rick Mugrige and Ward Cunningham (Prentice Hall, 2005), is still the ultimate resource on the subject. Plus, it gives a very good introduction to the acceptance-test tools Fit and FitNesse.

Here are URLs for tools mentioned in the main article:

- Selenium Core: http://seleniumhq.org/projects/core
- Selenium IDE: http://seleniumhq.org/projects/ide
- Selenium RC: http://seleniumhq.org/projects/remote-control
- Selenium Grid: http://selenium-grid.seleniumhq.org
- Abbot: http://abbot.sourceforge.net
- CubicTest: http://cubictest.openqa.org
- FitNesse: http://fitnesse.org
- HtmlUnit : http://htmlunit.sourceforge.net
- HttpUnit: http://httpunit.sourceforge.net
- Jemmy: http://jemmy.netbeans.org
- SWTBot: www.eclipse.org/swtbot
- WebDriver: http://code.google.com/p/webdriver

Hints for Practitioners

Mastering Selenium RC's API takes some time, but in most cases you can quickly find what you need. However, some problems require more than just examining the API.

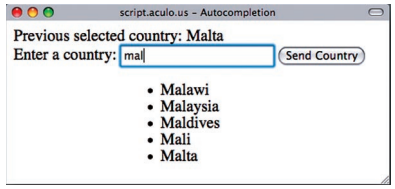
Users in the Driver's Seat

Users tend to quickly change their minds about tests, so programming can quickly appear to be a never-ending task. Instead of programming each test individually, programming fixtures for test orchestration

Table 2

Useful Selenese commands

Command	Purpose
Modifications	
type(locator, value)	Fill text fields, text areas, password fields, and so on.
select(selectLocator, optionLocator)	Select an option from a drop-down menu.
click(locator)	Push a button, check or uncheck a checkbox, select a radio button, or follow a link.
xxxAndWait	The same as above, but wait until the action has caused a navigation or refresh.
Checks	
verifyXXX(locator, pattern)	Verify whether an element matches the given pattern.
verifyXXXPresent(pattern)	Verify whether the element is present.
assert …	The same as verify, but terminate if verification fails.
waitForXXX	Wait until an element shows a certain quality or becomes present.



(a)

```
public class SeleniumAutocompletionTest {

    private Selenium browser;

    @Before
    public void setUp() {
        browser = new DefaultSelenium("localhost", 4444, "*firefox", "http://localhost:8080");
        browser.start();
    }

    @Test
    public void testAutocompletionCall() throws InterruptedException {
        browser.showContextualBanner();
        browser.open("/selenium-rc-ieee-example");

        assertEquals("script.aculo.us - Autocompletion", browser.getTitle());
        assertTrue(browser.isTextPresent("Previous selected country: null"));

        browser.typeKeys("id=country_input", "m");
        new Wait("Waiting for autocompletion box with 'Malta'") {
            public boolean until() {
                return browser.isElementPresent("xpath=//li[@id='Malta']");
            }
        };
        browser.click("xpath=//li[@id='Malta']");
        browser.click("send_country_button");
        browser.waitForPageToLoad("2000");

        assertEquals("script.aculo.us - Autocompletion", browser.getTitle());
        assertTrue(browser.isTextPresent("Previous selected country: Malta"));
    }

    @After
    public void tearDown() {
        browser.stop();
    }
}
```

(b)

Figure 1. Testing autocompletion of country codes using Selenium Remote Control for Java: (a) a very simple sample application and (b) a typical test with setup and teardown as well as access, modification, and verification. Remote Control methods have straightforward names, which makes life much easier. Fortunately, there are ready-to-use features for various kinds of waiting (the Wait **class in the middle and** waitForPageToLoad **further down), which we used to have to write ourselves when using more low-level tools.**

tools such as Fitnesse is a good idea. These fixtures group together actions such as filling out a complete form. They become building blocks for larger tests that don't require reprogramming but just editing and rescripting by users.

Speeding Things Up

Although tests with Selenium RC are much closer to the level of the user's experience than those with protocol-level tools such as HtmlUnit and HttpUnit, they come at a premium regarding time. Everything goes through the browser's JavaScript interpreter, which can make tests much slower than unit tests or GUI tests of desktop applications. Although Selenium Grid's purpose is to simulate an array of multiple users for load testing, you can use it to break a long test into smaller parts. These parts then execute in parallel on several machines instead of in sequence, which speeds things up.

Being Less Strict Helps

Because Selenium Core runs in the browser, it's subject to browser safety mechanisms such as the *same origin policy*, which prevents JavaScript code from spanning more than one site. Unless you want to test local applications only, turning off enforcement of that policy should be one of the first things you do.

Making Sure the Click Goes Through

The dynamic nature of Ajax applications is great for users but could become a nightmare for testers because keeping track of appearing, disappearing, hidden, and duplicate elements can get complicated. Although you could retrieve these elements programmatically, it's a good idea to assign unique IDs to application elements and refer to those IDs when writing tests. However, this is an option only if the application under test can be modified.

Easy Paths

If you can't use IDs, specifying *locators* becomes inevitable (you can use Javascript or XPath expressions for Dom traversal, as well as Cascading Style Sheets Selectors). Because these tend to be difficult to read and are repetitive, writing some helpers makes using locators much easier (see Figure 2).

Table 3

Three tools for Ajax acceptance tests

	Tool		
	Selenium	**HtmlUnit**	**WebDriver**
Stable release	Version 1.0	Version 2.5	Revision 964
Supported languages	C#, Java, Perl, PHP, Python, and Ruby	Java	Java
Supported browsers	Internet Explorer, Firefox, Safari, and Opera	None (works at the protocol level)	Internet Explorer and Firefox
Tools on top	CubicTest, Testmaker	WebTest	None
Ease of writing tests	Excellent	Sufficient	Good
Scope of test operations	Good	Sufficient	Excellent
Performance	Sufficient	Excellent	Good
Ease of installation	Good	Excellent	Sufficient
Functionality for regression and reporting	Not provided. JUnit, Fitnesse, and so on can provide that functionality by making calls to the tools' API.		

Selenium's key concept of the core running in the browser is a big step forward toward specifying above-protocol-level tests deserving the name "acceptance test." However, this approach entails all the problems of JavaScript, such as low speed and security issues. Hopefully, this issue will cease being a nuisance in future versions, which are expected to let users choose between executing tests by accessing Selenium Core inside the browser (as with the current version), using a simulated browser (using HtmlUnit or HttpUnit), or using WebDriver to externally control the browser. (Table 3 compares Selenium, HtmlUnit, and WebDriver.)

```
public void clickButtonByText(String buttonText) {
        browser.click("xpath=//input[@value='" + buttonText + "']");
    }
```

Figure 2. A helper for locating and then clicking a button by its text. Writing helpers can avoid long, repetitive locator expressions.

Andreas Bruns is a senior software engineer at C1 Workplace Solutions (C1 WPS), which focuses on consulting and managing in the areas of advanced software architecture, software transformation, and software engineering. Contact him at ab@c1-wps.de.

Andreas Kornstädt is a senior software architect at C1 WPS and carries out research at Stanford University's Center for Computer-Assisted Research in the Humanities. Contact him at ak@c1-wps.de or ak@ccrma.stanford.edu.

Dennis Wichmann is a software engineer at C1 WPS. Contact him at dw@c1-wps.de.