ERSelenium Framework

Screencast

http://www.wocommunity.org/podcasts/WOWODCW09-TDD.mov - 1 hour plus session on Test Driven Development with WebObjects presented by Denis Frolov on WOWODC 09. Shows how to use ERSelenium, jUnit, and Mockito with WebObjects.

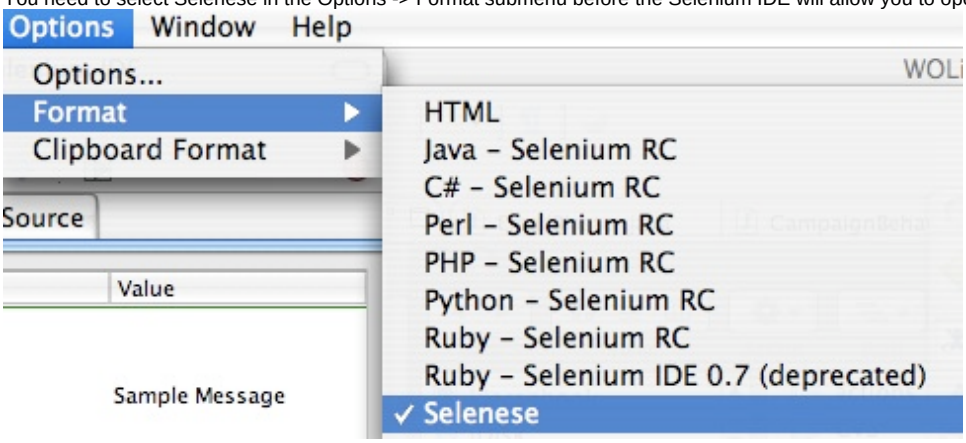
Quick Start

- Checkout the Project Wonder - the ERSelenium is in the Wonder/Frameworks/Misc folder.
- Examine test cases from Wonder/Examples/Misc/ERSeleniumExample/Resources/Selenium/main/ and from BugTracker in Eclipse text editor.
- Launch ERSeleniumExample and point your browser to SeleniumStartTesting Direct Action url (e.g. http://localhost/cgi-bin/WebObjects/ERSeleniumExample.woa/-42422/wa/SeleniumStartTesting).
- or Launch BugTracker and point your browser to SeleniumStartTesting Direct Action url (e.g. http://localhost/cgi-bin/WebObjects/BugTracker.woa/-42422/wa/SeleniumStartTesting).

You can also use Selenium IDE to create and edit tests:

- Launch FireFox and install Selenium IDE Firefox plugin.
- Using Selenium IDE format plugin installation instructions install Selenese-ide-plugin.js (it is in ERSelenium/Resources).
  Opening Tests in Selenium IDE

  You need to select Selenese in the Options -> Format submenu before the Selenium IDE will allow you to open tests in the Selenese format



- Play around with test cases from ERSeleniumExample/Resources/Selenium/ using Selenium IDE.

Overview and Usage Notes

ERSelenium provides several features for effective use of SeleniumCore with WebObjects applications including:

- Custom setup/teardown actions that can be run before/after each test.
- Base URL independence.
- Support of HTML and Selenese test formats.
- "On-the-fly" generation of test suites from the files in your project's source tree.
- Bookmarkable DirectAction url to run all tests (can be used for automated testing).
- Metacommands (special instructions specified in comments).

SeleniumCore is the powerful javascript toolkit for web applications "black-box" testing. It emulates different kinds of user actions such as: clicking the hyperlink, editing text in the input field, choosing item from the list and so on. See also: Selenium

Adding the ERSelenium framework dependency to an Application

You can add a ERSelenium test runner framework dependency to your application with Eclipse/WOLips:

1. Add the ERSelenium.framework to your project's Libraries dependency. See the tutorial: Add a Framework Dependency
2. (Optionally. Disabled by default. Use with caution). Enable the Selenium tests direct action url in production mode via the property:

   ```
   SeleniumTestsEnabled=true
   ```

   Debug output of ERSelenium can be enabled in Properties by:

```
log4j.logger.er.selenium = DEBUG
```

Other ERSelenium properties:

- **SeleniumTestsRoot="SomeOtherPath"** - change the tests location. By default ERSelenium will search for tests in "Resources/Selenium".
- **SeleniumReportPath="PathName"** - specified the path to the report file, which is created after the testing is done. "./Contents/Resources/" is the default value.

In your SeleniumTestsRoot folder (Resources/Selenium by default) you should create tests hierarchy. Tests are divided into groups, each group is located in its own folder. In each folder there should be a collection of test files, each in one of the formats, supported by ERSelenium. Example hierarchy:

```
./Resources
  ./Selenium
    ./registration
      ./successful.sel
      ./alreadyexists.sel
    ./shop
      ./buyitem.html
      ./notenoughmoney.html
      ./transfer.sel
```

You can use both standard HTML and wiki-like Selenese formats for writing tests although Selenese format is usually a preferred choice.

To run all tests point your browser to SeleniumStartTesting Direct Action:
http://baseurl/wa/SeleniumStartTesting

Example:
http://localhost/cgi-bin/WebObjects/SampleProject.woa/-42421/wa/SeleniumStartTesting

To run a specific group of tests, add "/TestGroupName":

```
http://baseurl/wa/SeleniumStartTesting/TestGroupName
http://127.0.0.1/cgi-bin/WebObjects/SampleProject.woa/-42421/wa/SeleniumStartTesting/registration
```

Some tips for writing tests for ERSelenium

- Don't use full URLs with open/openWindow commands (http://baseurl part will be added by ERSelenium):

  ```
  |open|/wa/EditPerson
  |open|/
  ```

- You can use setup/teardown methods. They should be implemented as direct actions in the separate class, which should be er.selenium.SeleniumAction-descendant. SeleniumAction class has some handy helper methods and automatically turns your selenium-related actions off when selenium is disabled in Properties. Here's the example of using selenium-related direct actions in the test (suppose that resetSessionAction() is defined in the class "Selenium"):

  ```
  |open|/wa/Selenium/resetSession
  ```

- You can use @repeat-@values-@done metacommands to execute specific part of the test with additional values edited in textboxes, e.g.:

```
@repeat
    ...some actions...
    @values user1 user2 user3
    |type|user|user0
    @values pass1 pass2 pass3
    |type|password|pass0
    ...some more actions...
  @done
```

The commands between @repeat and @done will be repeated several times, each time with new value in "user" and "password" input field. The values are seperated by spaces and if you have multiple @values lines, they all must have the same number of parameters. The @values section applies to the value of the next command.

Note: in Selenese format, lines that don't begin with "|" are treated as comments, so metacommands in the example above will be safely processed by Selenium IDE.

- Selenium IDE Firefox plugin and XPath Checker can be very handy for creating and editing Selenium tests. Selenium IDE Selenese source plugin with proper comments support resides in ERSelenium/Resources/selenese-ide-plugin.js.

Using basic flow control

ERSelenium comes with a preinstalled flowControl user-contributed extension that provides some basic flow control. You can use its commands as described in the flowControl documentation.

For example, consider a test page that displays an integer counter (in the format "Counter = *n*", where *n* is the current value of the counter), and has an "Increment" button with `id="increment"`. The following Selenese fragment would repeatedly press the button until the counter reached 10:

```
|while|!selenium.isTextPresent("Counter = 10");||
|click|increment||
|endWhile|||
```

Standalone runner

ERSelenium offers tests' developers several nice features - like automatic test suite generation, metacommands and URL independence. Unfortunately this leads to some troubles when trying to execute your tests with Selenium-RC. This is where StandaloneRunner can be very helpful.

Selenium-RC is essentially a selenium-server and a set of client libraries that you can use. You can write any kind of client application that sends particular commands to the server and receives back status codes. Among these commands are typical selenium commands ("click", "type" and so on) and several specific ("open specified browser"). Selenium-RC does a great job of preparing the browser profile (it turns off confirmation dialog boxes, clears cookies and so on) and gives a lot of other nice features. One of the most useful is the ability to execute tests from the command line. It's a typical task that is usually executed on build server on regular or per-commit basis.

The problem is that if you have your tests written with ERSelenium flavor, you won't be able to run them directly with Selenium-RC - as you won't even have a test suite file. But you can do this with StandaloneRunner. To execute the tests you must have Selenium-RC server running in the background. Note that it should be started in windowed mode (not via SSH) - i.e. not in the headless mode. Assuming that you have your application built, you should use the following commands:

```
YourApp.woa/YourApp <tests root folder> <application root url> <selenium-rc server host> <selenium-rc server
port> <browser type> \
     -DWOApplicationClass=er.selenium.rc.StandaloneRunner -Dfile.encoding=utf-8
```

This will execute the tests using the Selenium-RC server on the specified host and port with the specified browser. The <application root url> will be used as a root url for all urls that are used in tests. Tests from <tests root folder> will be executed (they will be searched for recursively).

Note, that two last arguments are essential: -DWOApplicationClass substitutes your application's application class with er.selenium.rc.StandaloneRunner, which will do all the testing and then exit without entering WOApplication's requests handling loop. The last argument ensures proper encoding.

Here is the real-world example commands:

```
cd /Library/WebObjects/Applications
./YourApp.woa/YourApp ./YourApp.woa/Contents/Resources/Selenium http://localhost/cgi-bin/WebObjects/YourApp.woa
localhost 4444 '*firefox' \
     -DWOApplicationClass=er.selenium.rc.StandaloneRunner -Dfile.encoding=utf-8
```

This will execute tests from **/Library/WebObjects/Applications/YourApp.woa/Contents/Resources/Selenium** using the http://localhost/cgi-bin/WebObjects/YourApp.woa as the root url in firefox browser using Selenium-RC server on **localhost** on its default port **4444**.

The sample of successful output is:

```
- test '/usr/local/wondercap2/dep/YourApp/14662/dep/dist/YourApp.woa/./Contents/Resources/Selenium/Commenting
/LoginViaCommenting.sel' PASSED
- test '/usr/local/wondercap2/dep/YourApp/14662/dep/dist/YourApp.woa/./Contents/Resources/Selenium/Commenting
/NotifyAboutReplies.sel' PASSED
- test '/usr/local/wondercap2/dep/YourApp/14662/dep/dist/YourApp.woa/./Contents/Resources/Selenium/Commenting
/SignUpViaCommenting.sel' PASSED
- test '/usr/local/wondercap2/dep/YourApp/14662/dep/dist/YourApp.woa/./Contents/Resources/Selenium/Commenting
/SuccessfulComment.sel' PASSED
```

All tests in tests root folder are always executed (even if there are failures). The executed command will fail if one of the tests fails.