Selenium IDE Guide for Testers

Tester's Guide to Selenium tests (Selenium IDE driven)

Getting started

There are three main pre-requisites you'll need to get started :

- **the Selenium IDE for Firefox**
  - This browser extension works on any Firefox-ready system and gives users the ability to create and run Selenium tests.
  - Download the Selenium IDE for Firefox.
- **the SeleniumTestCaseGenerator and associated project structure**
  - Check out the Selenium-SniffTests project via svn to convert and run java tests.
  - Latest version of the project structure.
  - See How the SeleniumTestCaseGenerator is implemented for more information.
- **the user-extension file** for advanced and eXo specific commands
  - This javascript file extends the basic Selenium commands for our specific use.
  - Latest version of the file.

Optional tools :

- **FireBug**
  - This browser extension helps analyze web content and eases locator definition.
  - Download FireBug for Firefox

Implementing Selenium test cases

The next paragraphs will help users implement Selenium test cases.

The basic steps are as follows :

1. **Creating a test case in .html using Selenium IDE**
2. **Running a test case in .html using Selenium IDE**
3. **Converting .html test cases to .java**
4. **Running a test case in .java using Maven**
5. **Improving the test case**

Creating a test case using Selenium IDE

1. After you install the Firefox plugin, a new option called "Selenium IDE" will be listed under the "Tools" dropdown in your browser. Click it to launch the Selenium IDE.
2. Include the user-extension.js file by choosing "Options..." from the "Options" dropdown in the IDE and browsing to add the Selenium Core

extension.
3. The Selenium IDE starts with the recording button pressed (click on it if it is not the case). Then, all you need to do is follow your manual test case in a Firefox window (clicking, typing, verifying) and the IDE will record your actions.
    - You can add verifications by right-clicking on the wanted object. The contextual menu will give a list of the basic available verifications.
    - Use Firebug to help identify locators (especially for dynamically created elements that could change at each session)
    - **Be sure to follow these basic principles : Selenium Test Recording Best Practices**
4. Save your new test case in the "...\selenium-snifftests\src\suite\org\exoplatform\...\selenium\" folder.

For more information on Selenium please see:

- Main Selenium Commands
- http://seleniumhq.org/docs/03_selenium_ide.html#chapter03-reference for a complete explanation of the Selenium IDE interface.
- Selenium Study Raw Notes for the eXo study on Selenium

Running a test case in .html using Selenium IDE

1. Open the test case by choosing "Open" from the "File" dropdown in the IDE and browsing to your working copy of the test. Its name should be listed in the Test Case sidebar on the right of the screen once you've loaded the test.
2. Enter the URL of your test instance into the "Base URL" navigation bar at the top of the screen.
3. You can adjust the rate at which the test runs by moving the slider bar between "Fast" and "Slow." However, all tests are (or should be) designed to work at full speed.
4. Press the "Play Current Test Case" button to begin the test.
    - You can pause the test by pressing the "Pause/Resume" button. Selenium will finish the currently-running command before pausing. You can resume by pressing the same button again.
    - Each command that completes successfully will be highlighted in green, while commands that fail will be highlighted in red. Most failed actions cause the test to stop.
    - You can choose a different starting point for the test by right-clicking any command and selecting the "Set / Clear Start Point"
    - Double-clicking any command will run only that command on Firefox's currently-displayed page.
5. After the test stops, check to make sure that nothing failed. If something did fail then you can read the log at the bottom of the page for more information regarding why.

A .html test case is considered "Done" when :

- it satisfies the following rules : Selenium Test Recording Best Practices
- it passes at least 5 times full speed

Converting .html test cases to .java using Maven

```
mvn generate-test-sources -Pselenium
```

Running a test case in .java using Maven

To convert and run all test cases from the target selenium folder

```
mvn install -Pselenium
```

Possible options :

```
to vary speed :
-Dselenium.speed=xxx
to run only a specific test :
-Dtest=Test*09.java
```

Running a test case on multiple browsers

```
-Dselenium.browser = iexploreproxy
-Dselenium.browser = safariproxy
-Dselenium.browser = googlechrome
-Dselenium.browser = firefox
```

Some browsers need special configuration to work with Selenium RC :
Safari : change proxy settings to focus only on http://localhost & port=8080
IE : change proxy settings to focus only on http://localhost & port=8080
FF :
Chrome :

Analyzing test results

Details on each test execution can be found in
"...\testsuite\selenium-snifftests\target\surefire-reports\"

Improving the test case

See Selenium Recording Tips & Tricks

Main Selenium Commands

Action commands :

Commands	Description	Arguments
check (locator)	Check a toggle-button (checkbox/radio)	• locator - an element locator
click (locator)	Clicks on a link, button, checkbox or radio button. If the click action causes a new page to load (like a link usually does), call waitForPageToLoad.	• locator - an element locator
clickAt (locator,coordString)	Clicks on a link, button, checkbox or radio button. If the click action causes a new page to load (like a link usually does), call waitForPageToLoad.	• locator - an element locator • coordString - specifies the x,y position (i.e. - 10,20) of the mouse event relative to the element returned by the locator
close ( )	Simulates the user clicking the "close" button in the titlebar of a popup window or tab.	
contextMenu (locator)	Simulates opening the context menu for the specified element (as might happen if the user "right-clicked" on the element).	• locator - an element locator
ContextMenuAt(locator,coordString)	Simulates opening the context menu for the specified element (as might happen if the user "right-clicked" on the element).	• locator - an element locator • coordString - specifies the x,y position (i.e. - 10,20) of the mouse event relative to the element returned by the locator.

deleteAllVisibleCookies ( )	Calls deleteCookie with recurse=true on all cookies visible to the current page. As noted on the documentation for deleteCookie, recurse=true can be much slower than simply deleting the cookies using a known domain/path.	
doubleClick (locator)	Double clicks on a link, button, checkbox or radio button. If the double click action causes a new page to load (like a link usually does), call waitForPageToLoad.	• locator - an element locator
doubleClickAt (locator,coordString)	Doubleclicks on a link, button, checkbox or radio button. If the action causes a new page to load (like a link usually does), call waitForPageToLoad.	• locator - an element locator • coordString - specifies the x,y position (i.e. - 10,20) of the mouse event relative to the element returned by the locator
dragAndDropToObject (locatorOfObjectToBeDragged,locatorOfDragDestinationObject)	Drags an element and drops it on another element	• locatorOfObjectToBeDragged - an element to be dragged • locatorOfDragDestinationObject - an element whose location (i.e., whose center-most pixel) will be the point where locatorOfObjectToBeDragged is dropped
echo (message)	Prints the specified message into the third table cell in your Selenese tables. Useful for debugging.	• message - the message to print
keyPress (locator,keySequence)	Simulates a user pressing a key (without releasing it yet).	• locator - an element locator • keySequence - Either be a string("\" followed by the numeric keycode of the key to be pressed, normally the ASCII value of that key), or a single character. For example: "w", "\119".
mouseDownRight (locator)	Simulates a user pressing the right mouse button (without releasing it yet) on the specified element.	• locator - an element locator
mouseOver (locator)	Simulates a user hovering a mouse over the specified element.	• locator - an element locator
open (url)	Opens an URL in the test frame. This accepts both relative and absolute URLs. The "open" command waits for the page to load before proceeding, ie. the "AndWait" suffix is implicit. Note: The URL must be on the same domain as the runner HTML due to security restrictions in the browser (Same Origin Policy). If you need to open an URL on another domain, use the Selenium Server to start a new browser session on that domain.	• url - the URL to open; may be relative or absoluter
pause (waitTime)	Wait for the specified amount of time (in milliseconds)	• waitTime - the amount of time to sleep (in milliseconds)

refresh ( )	Simulates the user clicking the "Refresh" button on their browser.	
select (selectLocator,optionLocator)	Select an option from a drop-down using an option locator. Option locators provide different ways of specifying options of an HTML Select element (e.g. for selecting a specific option, or for asserting that the selected option satisfies a specification). There are several forms of Select Option Locator. • label=labelPattern: matches options based on their labels, i.e. the visible text. (This is the default.) o label=regexp:^Oother • value=valuePattern: matches options based on their values. o value=other • id=id: matches options based on their ids. o id=option1 • index=index: matches an option based on its index (offset from zero). o index=2 If no option locator prefix is provided, the default behaviour is to match on label.	• selectLocator - an element locator identifying a drop-down menu • optionLocator - an option locator (a label by default)
selectFrame (locator)	Selects a frame within the current window. (You may invoke this command multiple times to select nested frames.) To select the parent frame, use "relative=parent" as a locator; to select the top frame, use "relative=top". You can also select a frame by its 0-based index number; select the first frame with "index=0", or the third frame with "index=2". You may also use a DOM expression to identify the frame you want directly, like this: dom=frames"main".frames"subframe"	• locator - an element locator identifying a frame or iframe
setSpeed (value)	Set execution speed (i.e., set the millisecond length of a delay which will follow each selenium operation). By default, there is no such delay, i.e., the delay is 0 milliseconds.	• value - the number of milliseconds to pause after operation
type (locator,value)	Sets the value of an input field, as though you typed it in. Can also be used to set the value of combo boxes, check boxes, etc. In these cases, value should be the value of the option selected, not the visible text.	• locator - an element locator • value - the value to type

typeKeys (locator,value)	Simulates keystroke events on the specified element, as though you typed the value key-by-key. This is a convenience method for calling keyDown, keyUp, keyPress for every character in the specified string; this is useful for dynamic UI widgets (like auto-completing combo boxes) that require explicit key events. Unlike the simple "type" command, which forces the specified value into the page directly, this command may or may not have any visible effect, even in cases where typing keys would normally have a visible effect. For example, if you use "typeKeys" on a form element, you may or may not see the results of what you typed in the field. In some cases, you may need to use the simple "type" command to set the value of the field and then the "typeKeys" command to send the keystroke events corresponding to what you just typed.	• locator - an element locator • value - the value to type.
windowMaximize ( )	Resize currently selected window to take up the entire screen	
componentExoContextMenu(locator)	Does a right click on an object ans calls the eXo context menu	• locator - an element locator

waitFor commands :

Commands	Description
waitForElementPresent	Waits that the specified element is somewhere on the page. Useful before all clickAt commands.
waitForText	Waits that the specified text (argument 2) is at the specified location (argument 1). Useful before all clickAt commands.
waitForTextPresent	= Waits that the specified text is somewhere on the page. Useful before all clickAt commands.
waitForPageToLoad (timeout)	Waits for a new page to load. You can use this command instead of the "AndWait" suffixes, "clickAndWait", "selectAndWait", "typeAndWait" etc. (which are only available in the JS API). Selenium constantly keeps track of new pages loading, and sets a "newPageLoaded" flag when it first notices a page load. Running any other Selenium command after turns the flag to false. Hence, if you want to wait for a page to load, you must wait immediately after a Selenium command that caused a page-load.

Verification commands :

Commands	Description
verifyElementPresent	true if the pattern matches the text, false otherwise Verifies that the specified text pattern appears somewhere on the rendered page shown to the user.
verifyText	Verifies that the specified text (argument 2) is at the specified location (argument 1).
verifyTextPresent	Verifies that the specified text is somewhere on the page.

All Selenium commands (supported by converter)

Not all Selenium commands have been implemented in the generator. If a command is not translated to java, a warning will appear in the command console. Ask the Release Team for help in implementing a new command.

Here is a list of supported commands

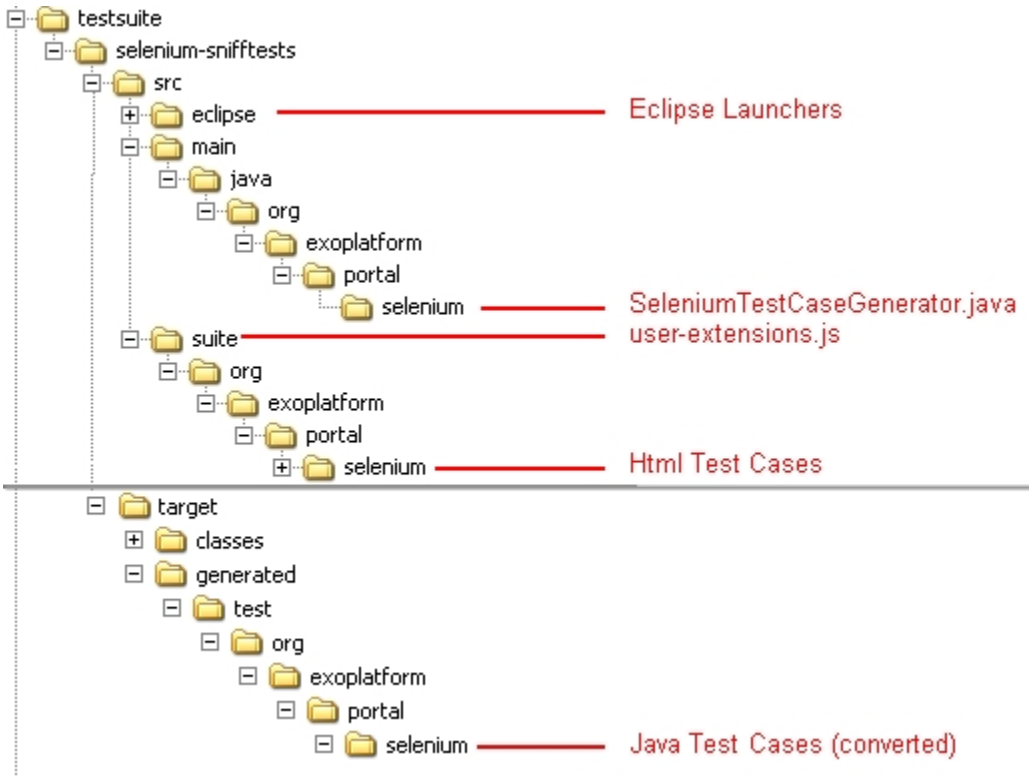
Do's and Don'ts

- **Do** modify the html with Selenium IDE, save to ".../testsuite/selenium-snifftests/src/suite/org/exoplatform/.../selenium", and re-run the generator.
- **Do not** modify the java tests manually (they will be overwritten after next test generation)

How the SeleniumTestCaseGenerator is implemented

Tree View

Here is the complete tree view for a Selenium testsuite with the main files :
(example for Portal)



- The .html scripts are saved in the ".../testsuite/selenium-snifftests/src/suite/org/exoplatform/.../selenium" file.
- The user-extension.js file used by the generator is found in ".../testsuite/selenium-snifftests/src/suite".
  - *you can also use this for the Selenium Core user-extension.*
- Once the tests are converted ("Generated") to .java, the ".../testsuite/selenium-snifftests/target" file will be added.
- Surefire reports (test execution logs) are stored in ".../testsuite/selenium-snifftests/target/surefire-reports".

Selenium Day Presentation

You need flash player installed to preview ppt and pdf files