Software Engineering Testing   Practical Session: Selenium

Selenium is one of the most well-known testing frameworks in the world that is in use. It is an open source project that allows testers and developers alike to develop functional tests to drive the browser. It can be used to record workflows so that developers can prevent future regressions of code. Selenium can work on any browser that supports JavaScript, since Selenium has been built using JavaScript.

Task 1: Recording your first test with Selenium IDE

1. When in record mode, navigate to http://book.theautomatedtester.co.uk/chapter1
2. On the Web Applicaton do the following:
    - Click on the radio button.
    - Select another value from the drop-down box, for example, **Selenium RC**.
3. Your test has now been recorded, so Click the play button
4. Save the test case as Task1

As we can see Selenium IDE has tried to apply the first rule of test automaton by specifying the open command. It has set the starting point of the test, in this case /chapter1, and then it began stepping through the workflow that we want to record.

Exercise 1:
Record a Test Script to open http://courseweb.sliit.lk/ and click the link of "General Support".
Save the test case as Exercise1

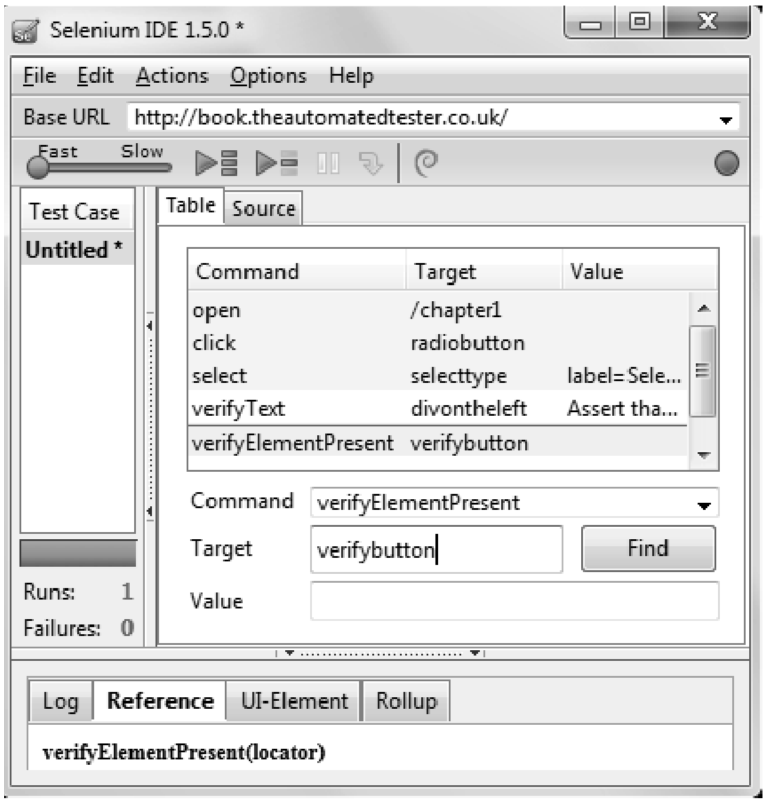
Updating a test to assert items are on the page

There are two mechanisms for validating elements available on the application under test.
The first is **assert**; this allows the test to check if the element is on the page. If it is not available then the test will stop on the step that failed. The second is **verify**; this also

allows the test to check the element is on the page, but if it isn't then the test will carry on executing.

Task 2: Verify items on the page

1. Open the IDE so that we can start recording.
2. Navigate to http://book.theautomatedtester.co.uk/chapter1.
3. Select Selenium Grid from the drop-down box.
4. Change the Select to Selenium Grid.
5. Verify that Assert that this text is on the page text is mentioned on the right hand side of the drop-down box, by right-clicking on the text and selecting Verify TextPresent Assert that this text is on the page.
6. Verify that the button is on the page. You will need to add a new command for
7. verifyElementPresent with the target verifybutton in Selenium IDE.
8. Now that you have completed the previous steps, your Selenium IDE should look like the following screenshot:
9. Save the test case as Task2



Base URL: http://book.theautomatedtester.co.uk/

Command	Target	Value
open	/chapter1	
click	radiobutton	
select	selecttype	label=Sele...
verifyText	divontheleft	Assert tha...
verifyElementPresent	verifybutton	

Command: verifyElementPresent
Target: verifybutton
Value:

verifyElementPresent(locator)

Exercise 2:

Record a Test Script to do followings.

- open http://www.sliit.lk/
- Click on About Sliit link
- Assert the Text "Sri Lanka Institute of Information Technology"
- Verify the SLIIT Logo is present

Save the test case as Exercise2

Exercise 3:

- Open the above recorded script and do followings.
- Select the command "assertText"
- Change the value to "Sri Lanka Institute"
- Re run the test case and see what will happen

Save the test case as Exercise2

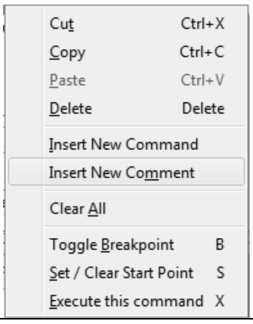
Exercise 4:

- Open the above recorded script and do followings.
- Select the command "assertText"
- Change the command to "verifyText"
- Re run the test case and see what will happen

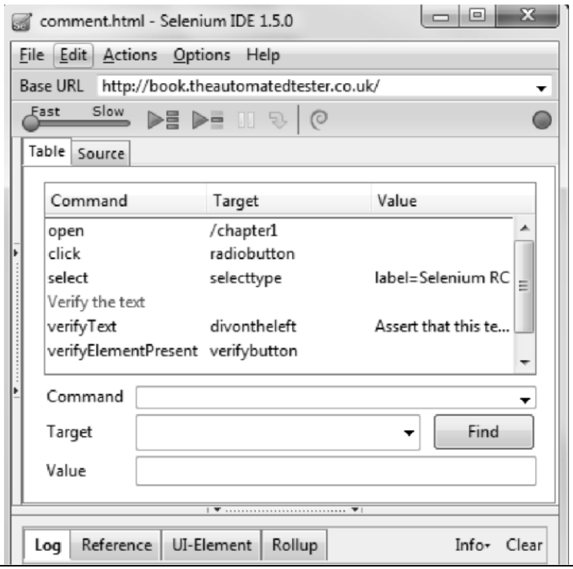
Save the test case as Exercise4

Task 3: adding Selenium IDE comments

1. In the test that was created earlier, right-click on a step. For example, the verify step.
2. The Selenium IDE context menu will be visible as shown in the following screenshot:



3. Click on Insert New Comment. A space will appear between the Selenium commands.
4. Click on the Command textbox and enter in a comment so that you can use it for future maintenance. It will look like the following screenshot:



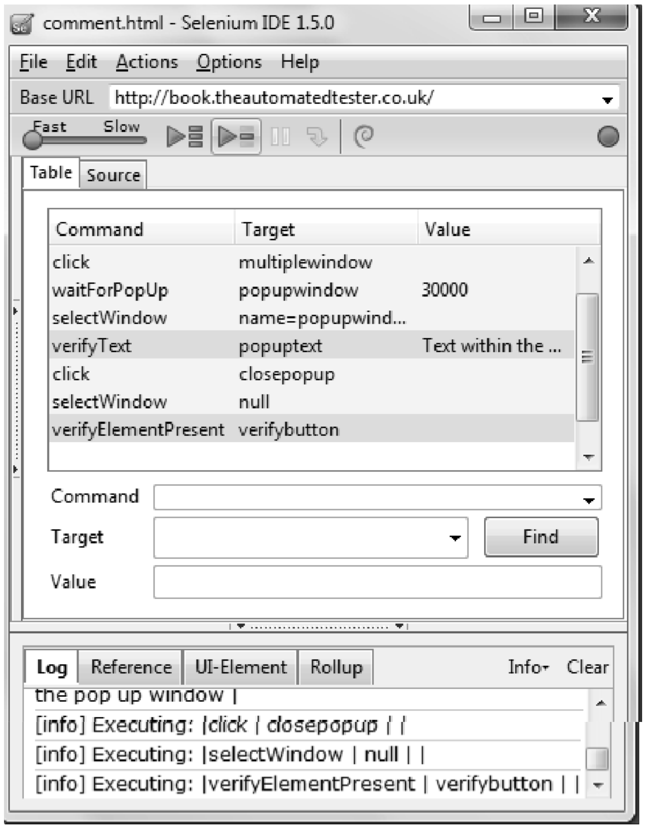
Base URL: http://book.theautomatedtester.co.uk/

Command	Target	Value
open	/chapter1	
click	radiobutton	
select	selecttype	label=Selenium RC
Verify the text		
verifyText	divontheleft	Assert that this te...
verifyElementPresent	verifybutton	

Task 4: Multiplying windows

Web applications unfortunately do not live in one window of your browser. An example of this could be a site that shows reports. Most reports would have their own window so that people can easily move between them.

1. Open up Selenium IDE and go to the **Chapter 1** page on the site.
2. Click on one of the elements on the page that has the text **Click this link to launch another window**. This will cause a small window to appear.
3. Verify the text in the popup by right-clicking and selecting **VerifyText id=popup text within the popup window**.
4. Once the window has loaded, click on the **Close the Window** text inside it.
5. Add a verify command for an element on the page. Your test should now look like the following screenshot:
Save the test case as Task4


comment.html - Selenium IDE 1.5.0
File Edit Actions Options Help
Base URL http://book.theautomatedtester.co.uk/
Fast Slow
Table Source
Find
Log Reference UI-Element Rollup Info Clear

Command	Target	Value
click	multiplewindow	
waitForPopUp	popupwindow	30000
selectWindow	name=popupwind...	
verifyText	popuptext	Text within the ...
click	closepopup	
selectWindow	null	
verifyElementPresent	verifybutton	

Command

Target

Value

```
the pop up window |
[info] Executing: |click | closepopup | |
[info] Executing: |selectWindow | null | |
[info] Executing: |verifyElementPresent | verifybutton | |
```

In the test script we can see that it has clicked on the item to load the new window and then has inserted a **waitForPopUp**. This is so that your test knows that it has to wait for a web server to handle the request and the browser to render the page. Any commands that require a page to load from a web server will have a **waitFor** command.

The next command is the **selectWindow** command. This command tells Selenium IDE that it will need to switch context to the window, called **popupwindow**, and will execute all the commands that follow in that window unless told otherwise by a later command.

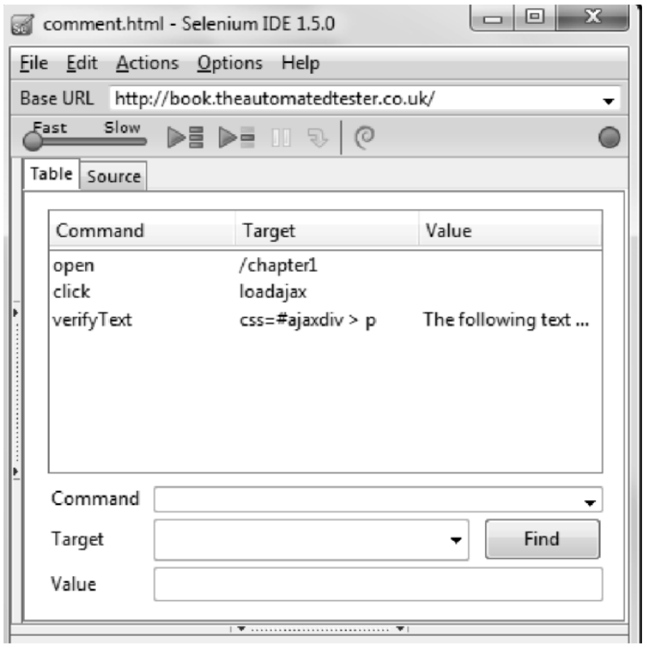
Once the test has finished with the popup window, it will need to return to the parent window from where it started. To do this we need to specify null as the window. This will force the **selectWindow** to move the context of the test back to its parent window.

Selenium tests against AJAX applications

Web applications today are being designed in such a way that they appear the same as desktop applications. Web developers are accomplishing this by using AJAX within their web applications. **AJAX** stands for **Asynchronous JavaScript And XML** due to the fact that it relies on JavaScript creating asynchronous calls and then returning XML with the data that the user or application requires to carry on. AJAX does not rely on XML anymore, as more and more people move over **JSON**, **JavaScript Object Nota□ on** , which is more lightweight in the way that it transfers the data. It does not rely on the extra overhead of opening and closing tags that is needed to create valid XML.

Task 5: working on pages with AJAX

1. Start up Selenium IDE and make sure that the Record button is pressed.
2. Navigate to http://book.theautomatedtester.co.uk/chapter1.
3. Click on the text that says **Click this link to load a page with AJAX**.
4. Verify the text that appears on your screen. Your test should look like the following screenshot:

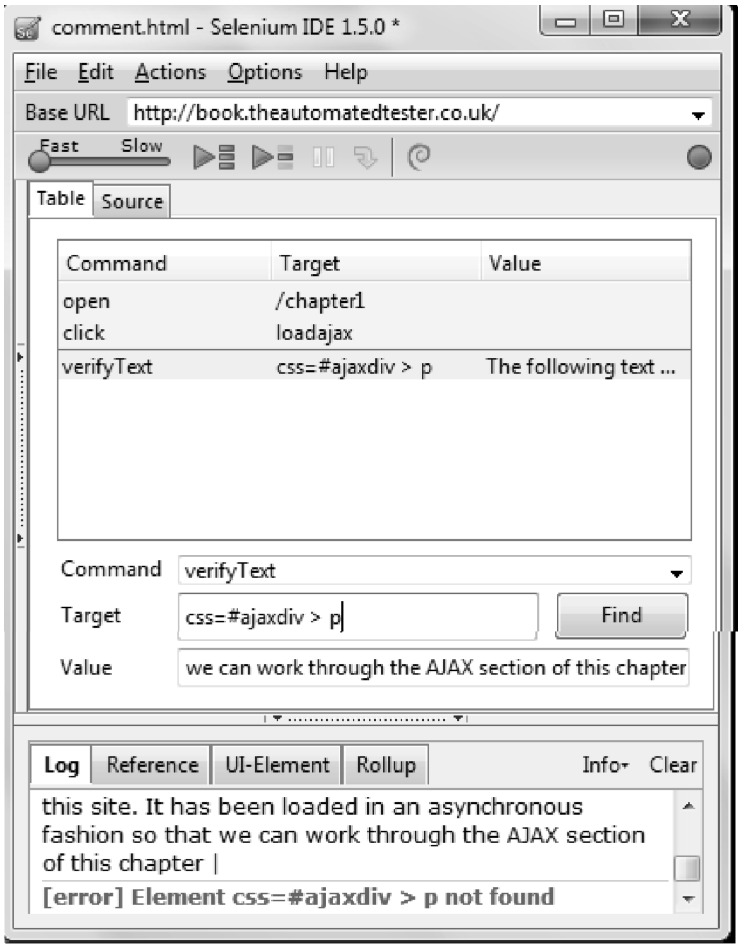

comment.html - Selenium IDE 1.5.0
File Edit Actions Options Help
Fast Slow
Table Source
Find

Base URL: http://book.theautomatedtester.co.uk/

Command	Target	Value
open	/chapter1	
click	loadajax	
verifyText	css=#ajaxdiv > p	The following text ...

Command:
Target:
Value:

5. Run the test that you have created. When it has finished running it should look like the following screenshot:


comment.html - Selenium IDE 1.5.0 *
File Edit Actions Options Help
Base URL http://book.theautomatedtester.co.uk/
Fast Slow
Table Source
Log Reference UI-Element Rollup Info▾ Clear
Find

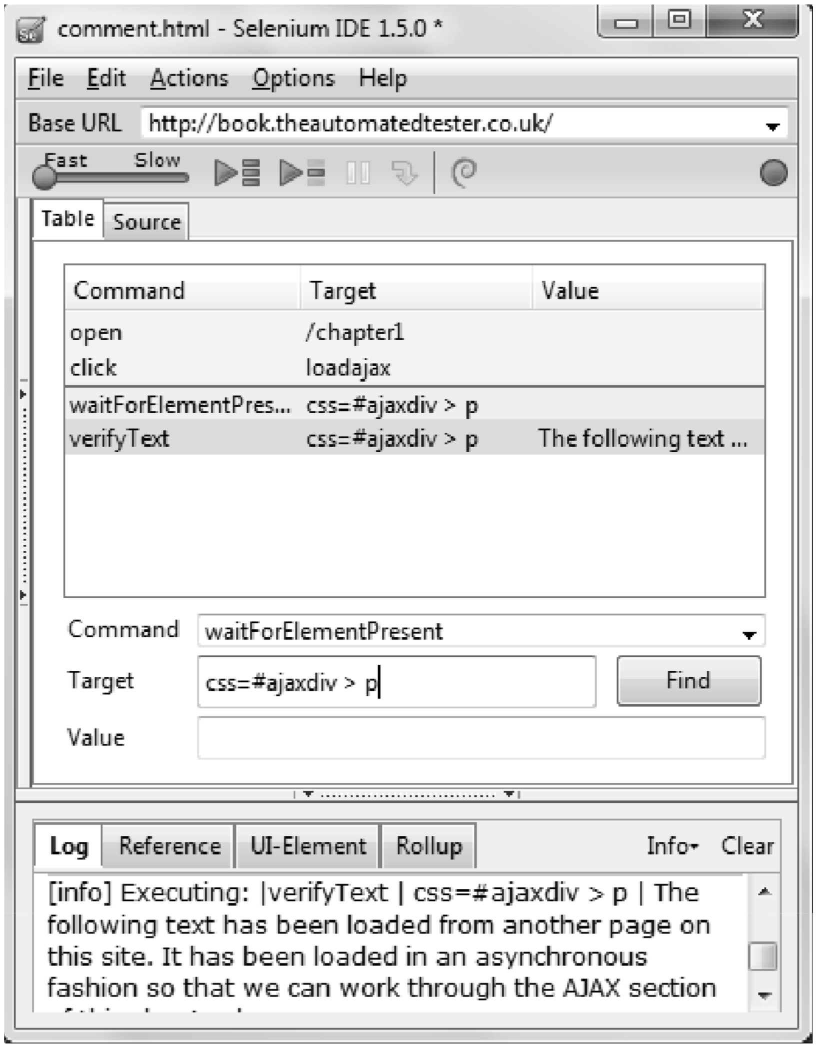
Command	Target	Value
open	/chapter1	
click	loadajax	
verifyText	css=#ajaxdiv > p	The following text ...

Command: verifyText
Target: css=#ajaxdiv > p
Value: we can work through the AJAX section of this chapter

```
this site. It has been loaded in an asynchronous
fashion so that we can work through the AJAX section
of this chapter |
[error] Element css=#ajaxdiv > p not found
```

Have a look at the page that you are working against. Can you see the text that the test is expecting? You should see it, so why has this test failed? The test has failed because when the test reached that point, the element containing the text was not loaded into the DOM. This is because it was being requested and rendered from the web server into the browser. To remedy this issue, we will need to add a new command to our test so that our tests pass in the future:

1. Right-click on the step that failed so the Selenium IDE context menu appears.
2. Click on **Insert New Command**.
3. In the **Command** select box, type **waitForElementPresent** or select it from the dropdown menu.
4. In the **Target** box add the target that is used in the **verifyText** command.
5. Run the test again and it should pass this time:


comment.html - Selenium IDE 1.5.0 *
File Edit Actions Options Help
Base URL http://book.theautomatedtester.co.uk/
Fast Slow
Table Source
Find
Log Reference UI-Element Rollup Info Clear

Command	Target	Value
open	/chapter1	
click	loadajax	
waitForElementPres...	css=#ajaxdiv > p	
verifyText	css=#ajaxdiv > p	The following text ...

Command: waitForElementPresent
Target: css=#ajaxdiv > p
Value:

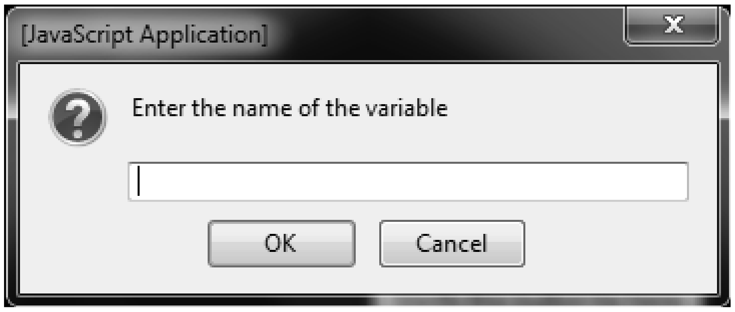
```
[info] Executing: |verifyText | css=#ajaxdiv > p | The
following text has been loaded from another page on
this site. It has been loaded in an asynchronous
fashion so that we can work through the AJAX section
```

Selenium does not implicitly wait for the item that it needs to interact with, so it is seen as good practice to wait for the item you need to work with then interact with it. The waitFor commands will timeout a timer 30 seconds by default but if you need it to wait longer you can specify the tests by using the setTimeout command. This will set the timeout value that the tests will use in future commands.
If need be you can change the default wait if you go to **Options | Options** and then on the **General** tab and under **Default timeout** value of recorded command in milliseconds (30s = 30000ms) change it to what you want. Remember that there are 1000 milliseconds in a second.

Task 8: Storing information from the page in the test

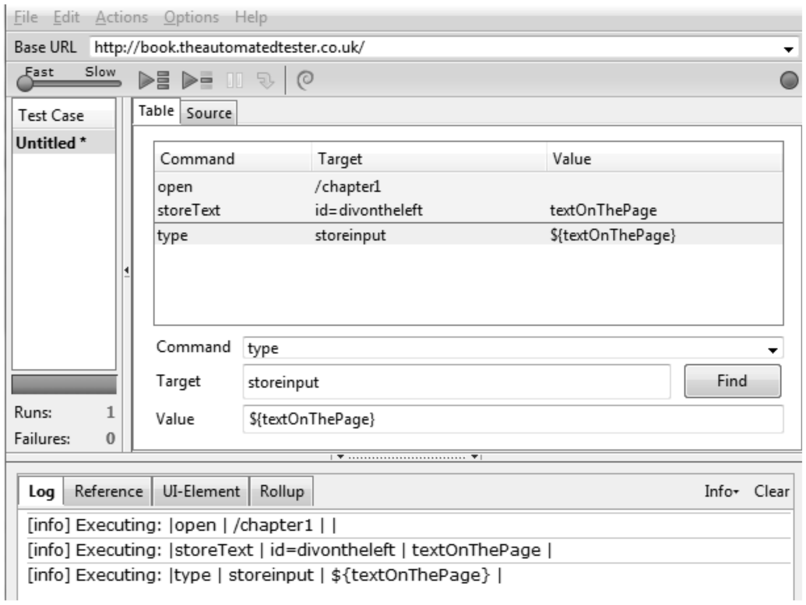
Sometimes there is a need to store elements that are on the page to be used later in a test. This could be that your test needs to pick a date that is on the page and use it later so that you do not need to hardcode values into your test. Once the element has been stored you will be able to use it again by requesting it from a JavaScript dictionary that Selenium keeps track of. To use the variable it will take one of the following two formats: it can look like ${variableName} or storedVars['variableName']. I prefer the storedVars format as it follows

1. Open up Selenium IDE and switch off the Record button.
2. Navigate to http://book.theautomatedtester.co.uk/chapter1.
3. Right-click on the text **Assert that this text is on the page** and go to the storeText command in the context menu and click on it.
4. A dialog will appear as shown in the following screenshot. Enter the name of a variable that you want to use. I have used **textOnThePage** as the name of my variable.


[JavaScript Application]

Enter the name of the variable

5. Click on the row below the storeText command in Selenium IDE.
6. Type type into the **Command** textbox.
7. Type storeinput into the **Target** box.
8. Type ${textOnThePage} into the **Value** box.
9. Run the test. It should look like the following screenshot:


File Edit Actions Options Help
Fast Slow
Test Case
Untitled *
Table Source
Find
Log Reference UI-Element Rollup Info Clear

Base URL http://book.theautomatedtester.co.uk/

Command	Target	Value
open	/chapter1	
storeText	id=divontheleft	textOnThePage
type	storeinput	${textOnThePage}

Command type
Target storeinput
Value ${textOnThePage}

Runs: 1
Failures: 0

```
[info] Executing: |open | /chapter1 | |
[info] Executing: |storeText | id=divontheleft | textOnThePage |
[info] Executing: |type | storeinput | ${textOnThePage} |
```

Once your test has completed running you will see that it has placed **Assert that this text is on the page** into the textbox.