

Test Automation & Advanced Selenium

Lesson 3: Working With Selenium IDE



Lesson Objectives



- Selenium IDE – An Introduction
- Installation of Selenium IDE
- Opening the Selenium IDE
- Components of Selenium IDE
- Introduction to Selenium IDE Commands – “Selenese”
- Capabilities of Selenium IDE Commands
- Types of Selenium IDE Commands
- Selenium IDE Commands – Some Common Commands
- Understanding Element Locators in Selenium IDE
- Locators in Selenium
- Locating Elements by CSS Selectors
- Locating Elements by DOM
- Introduction to XPath



Lesson Objectives



- Types of XPath
- Locating Elements by XPath
- Store Commands
- Introduction to Alert Selenium IDE Commands
- Introduction to Confirmation Selenium IDE Commands
- Introduction to Debugging in Selenium IDE
- Using Breakpoints in Test Case
- Using Startpoint in Test Case
- Using Firebug to identify object
- Create Script Using Selenium IDE
- Exporting scripts to multiple languages and Formats



3.1: Working With Selenium IDE

Selenium IDE – An Introduction

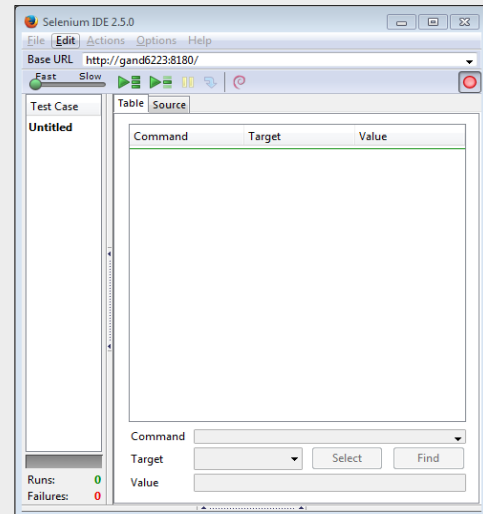


- Selenium IDE is an integrated development environment for Selenium tests
- It is implemented as a Firefox extension, and allows you to record, edit, and replay the web test in Firefox
- Selenium provides a record/playback tool for authoring tests without learning a test scripting language (Selenium IDE)
- Selenium IDE, fully-featured Integrated Development Environment (IDE)
- Records user interactions with the web browser and play back to test for errors
- Effortless to install and easy to learn
- Using Selenium IDE is a great option available to testers to get started with writing test and group them together to build the Test Suit
- The recorded tests can be exported to many programming languages so that we can tweak them and put them in the testing framework
- The test cases and test suites can be replayed back to check the verifications and validations

3.1: Working With Selenium IDE

Installation of Selenium IDE

- Download Selenium IDE
- Copy Selenium IDE file to extensions folder of Mozilla Firefox
- To facilitate opening of .xpi file for selenium you can drag and drop the .xpi file on the Mozilla browser to install it and get it in the add-ons of the browser

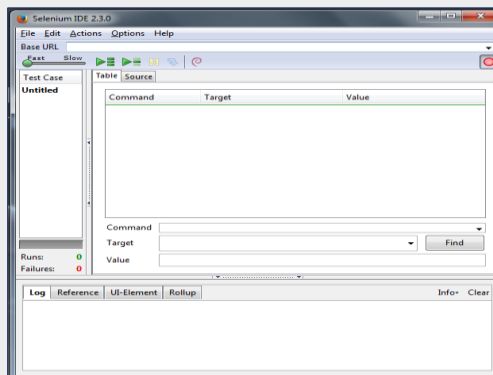


3.1: Working With Selenium IDE

Opening the Selenium IDE

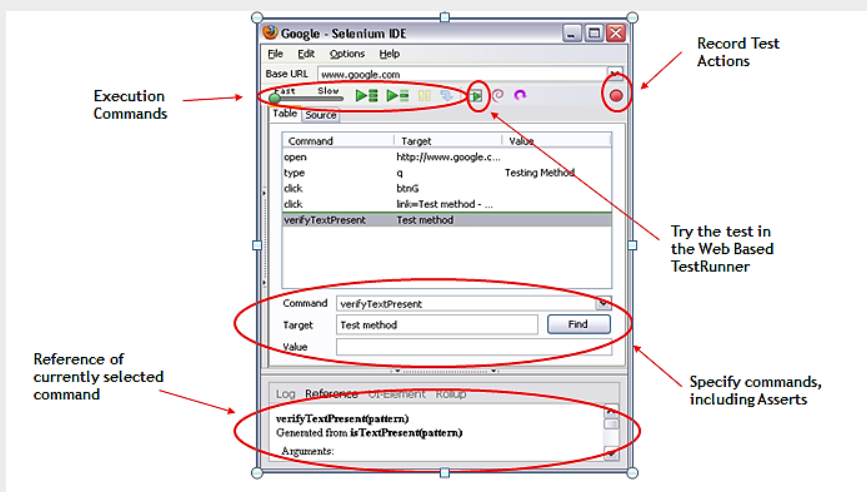


- To run the Selenium-IDE, simply select it from the Firefox Web Developer option
- It opens as follows with an empty script-editing window and a menu for loading, or creating new test cases



3.1: Working With Selenium IDE

Components of Selenium IDE

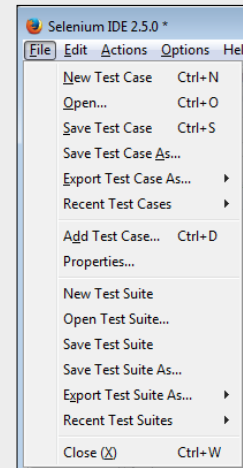


3.1: Working With Selenium IDE

Components of Selenium IDE (Cont.)

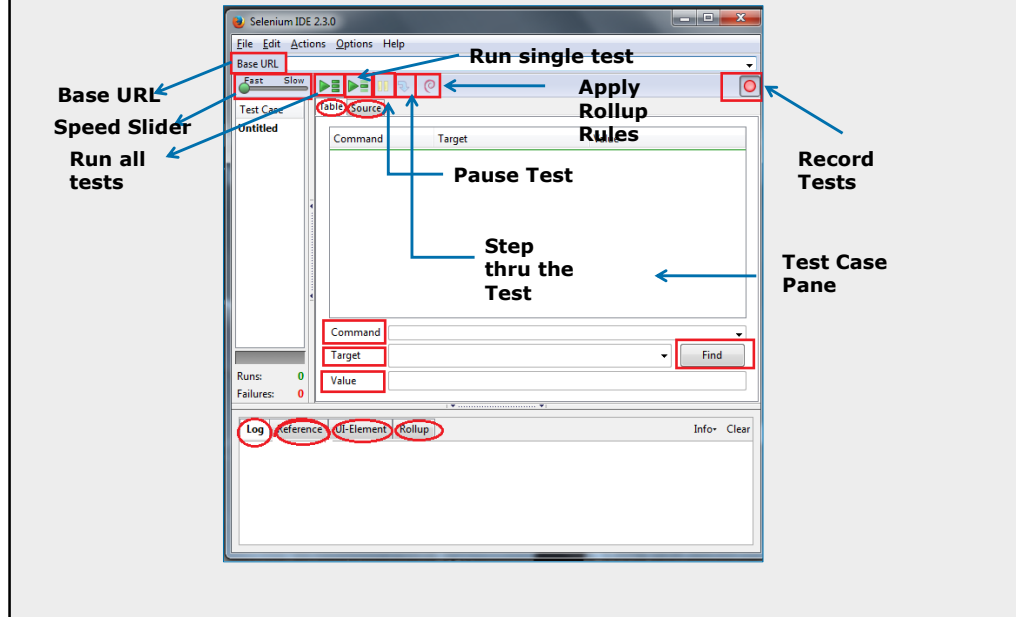


- Menu Bar
- Positioned at the upper most of the Selenium IDE window
- File Menu
 - Allows user to:
 - Create new test case, open existing test case, save the current test case
 - Export Test Case As option exports and converts only the currently opened Selenium IDE test case
 - Export Test Case As and Export Test Suite As in any of the associated programming language compatible with Selenium RC and WebDriver
 - Export Test Suite As option exports and converts all the test cases associated with the currently opened IDE test suite
 - Close the test case



3.1: Working With Selenium IDE

Components of Selenium IDE (Cont.)

**Components of Selenium IDE:**

1. **Base URL**: This is the URL that the test will start at. All open commands will be relative to the Base URL unless a full path is inserted in the open command.
2. **Speed Slider**: This is the slider under the Fast and Slow labels on the screen.
3. **Run all tests**: Run all the tests in the IDE.
4. **Run single test**: Run a single test in the IDE.
5. **Pause Test**: Pause a test that is currently running.
6. **Step thru the test**: Step through the test once it has paused.
7. **Apply Rollup Rules**: This advanced feature allows repetitive sequences of Selenium commands to be grouped into a single action.
8. **Record**: Records the user's browser actions.
9. **Test Case Pane**: Your script is displayed in the test case pane. It has two tabs, one for displaying the command and their parameters in a readable "table" format.
10. **Source**: Source displays the test case in the native format in which the file will be stored. By default, this is HTML although it can be changed to a programming language such as Java or C#, or a scripting language like Python.
11. **Target**: The Target textbox allows you to input the location of the element that you want to work against.
12. **Find**: The Find button, once the target box is populated, can be clicked to highlight the element on the page.
13. **Value**: The Value textbox is where you place the value that needs to change. For example, if you want your test to type in an input box on the web page, you would put what you want it to type in the value box.
14. **Log**: When you run your test case, error messages and information messages showing the progress are displayed in this pane automatically, even if you do not first select the Log tab. These messages are often useful for test case debugging.

3.1: Working With Selenium IDE

Components of Selenium IDE (Cont.)



- Test Case Pane : Script is displayed in the test case pane
- It has two tabs:
 - Displays the command and their parameters in a readable “Table” format
 - “Source” displays the test case in the native format in which the file will be stored
 - By default, this is HTML although it can be changed to a programming language such as Java or C#, or a scripting language like Python

Command	Target	Value
open	/	
waitForPageToLoad		
clickAndWait	xpath=id('menu_download')/a	
assertTitle	Downloads	
verifyText	xpath=id('mainContent')/h2	Downloads

3.1: Working With Selenium IDE

Components of Selenium IDE (Cont.)



- Command, Target and Value

- Command displays the currently selected command along with its parameters
- Target displays unique tag value for the selected field
- Value contains the data in case of text box fields

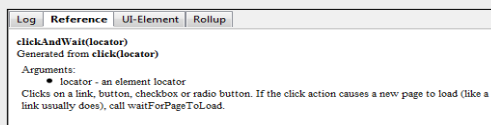
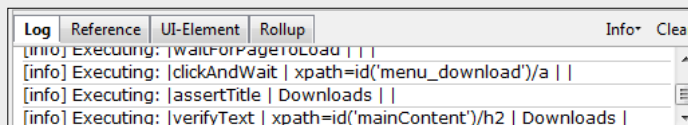
Command	clickAndWait	
Target	<input type="text" value="xpath=id('menu_download')/a"/>	<input type="button" value="Find"/>
Value	<input type="text"/>	

3.1: Working With Selenium IDE

Components of Selenium IDE (Cont.)



- Log/Reference/UI-Element/Rollup Pane - Bottom pane is used
 - Log: Error messages and information messages showing the progress are displayed automatically
 - Reference: Displays documentation on the current command in table mode
 - UI-Element and Rollup: Detailed information on these two panes (which cover advanced features) can be found in the UI-Element Documentation on the Help menu of Selenium-IDE

**Log/Reference/UI-Element/Rollup Pane**

The bottom pane is used for four different functions—Log, Reference, UI-Element, and Rollup—depending on which tab is selected.

Log

When you run your test case, error messages and information messages showing the progress are displayed in this pane automatically, even if you do not first select the Log tab. These messages are often useful for test case debugging. Notice the Clear button for clearing the Log. Also notice the Info button is a drop-down allowing selection of different levels of information to log.

Reference

The Reference tab is the default selection whenever you are entering or modifying Selenese commands and parameters in Table mode. In Table mode, the Reference pane will display documentation on the current command. When entering or modifying commands, whether from Table or Source mode, it is critically important to ensure that the parameters specified in the Target and Value fields match those specified in the parameter list in the Reference pane. The number of parameters provided must match the number specified, the order of parameters provided must match the order specified, and the type of parameters provided must match the type specified. If there is a mismatch in any of these three areas, the command will not run correctly.

While the Reference tab is invaluable as a quick reference, it is still often necessary to consult the Selenium Reference document.

Demo – First Selenium IDE Test Case



- Recording our first Test
- Running the Test
- Inserting commands

3.2: Working with Selenium IDE Commands



Introduction to Selenium IDE Commands – “Selenese”

- Selenium commands, often called as “Selenese”, are the set of commands that run your tests
- A sequence of these commands is a test script
- Selenium provides a rich set of commands for fully testing your web-app in virtually any way you can imagine
- These commands essentially create a testing language
- Selenese is essentially just a language which is nothing but the syntax and not dependent upon any language like C#, Java etc.
- Selenese commands can have up to a maximum of two parameters: target and value

3.2: Working with Selenium IDE Commands



Capabilities of Selenium IDE Commands

- With help of Selenese one can :
 - Test the existence of UI elements based on their HTML tags
 - Test for specific content
 - Test for broken links, input fields, selection list options, submitting forms, and table data among other things
- In addition Selenese supports testing of:
 - Window size
 - Mouse position
 - Alerts
 - Ajax functionality
 - Pop up windows
 - Event handling
 - And many other web-application features

3.2: Working with Selenium IDE Commands



Types of Selenium IDE Commands

Type	Description
Actions	<p>These are the commands that changes the state of the application by directly interacting with the page elements.</p> <p>Example: Click the link, Select the option, Type text</p> <p>If an Action fails, or has an error, the execution of the current test is stopped.</p> <p>The "AndWait" suffix is used while calling the action. For example "clickAndWait", this suffix instructs Selenium that it should wait for a new page to load.</p>
Accessors	<p>These are commands that allow you to examine the state of the application and store results in variables, e.g. "storeTitle".</p>

Types of Selenium IDE Commands:**Action:**

Used to change the state of the AUT(Application under Test)like click on some link, type a value in edit box, select some options on the page, select a value from drop down etc.

When action is performed on AUT the test will fail if the action is not achieved.

Accessor:

This command check the state of the application and save the application state in some variable. It can be used for automatic generation of assertions.

Assertions:

Are very similar to checkpoint in UFT/QTP. Assertion verifies the state of the application matches it with the expected state and generates the True/False result.

3.2: Working with Selenium IDE Commands



Types of Selenium IDE Commands (Cont.)

Type	Description
Assertions	<p>They are like Accessors, but they verify that the state of the application conforms to what is expected.</p> <p>Examples: "make sure the page title is something" and "verify that this radiobutton is selected".</p> <p>Three types of assertions</p> <p>Assert: When an "assert" fails, the test is aborted. For example "assertText"</p> <p>Verify: When a "verify" fails, the test will continue execution, logging the failure. For example "verifyText"</p> <p>WaitFor: Before proceeding to the next command, "waitFor" commands will first wait for a certain condition to become true.</p> <p>Step passes - If the condition becomes true within the waiting period (30 Seconds).</p> <p>Step fails - If the condition does not become true. Failure is logged, and test execution proceeds to the next command.</p>

3.2: Working with Selenium IDE Commands



Selenium IDE Commands – Some Common Commands

Command	Description
Open	It opens up the page using given URL
click/clickAndWait	It performs click operation and optionally waits for a new page to load
verifyTitle/assertTitle	It verifies an expected page title
verifyTextPresent	It verifies that the expected text is present on the page
verifyElementPresent	It verifies an expected UI element, as defined by its HTML tag, is present on the page
verifyText	It verifies that the expected text along with its HTML tag are present on the page
verifyTable	It can be used to verify the expected content on the table
waitForPageToLoad	It pauses execution until an expected new page loads. Called automatically when clickAndWait is used
waitForElementPresent	It pauses the execution until the expected UI is present on the web page

Demo – Validations in Selenium IDE Test Case



- Using Assert and Verify in the test
- Using waitFor in the test

3.3: Introduction to Element Locators in Selenium IDE



Understanding Element Locators in Selenium IDE

- The “Locators” informs Selenium IDE about which GUI elements it is supposed to operate on
- Identification of correct GUI elements is a prerequisite to create an automation script
- Identifying the GUI element on a web page accurately is more difficult it sounds
- Sometimes we end up working on wrong GUI element or no elements at all
- Therefore, Selenium facilitates us with number of locators to precisely locate a GUI element on the web page

3.3: Introduction to Element Locators in Selenium IDE

Locators in Selenium



- The different types of Locators are given below :
 - ID
 - Name
 - Link Text
- CSS Selector
 - Tag and ID
 - Tag and Class
 - Tag and Attribute
 - Tag, Class, and attribute
 - Inner Text
- DOM (Document Object Model)
 - getElementById
 - getElementsByName
 - dom:name
 - dom:index
- XPath

3.3: Introduction to Element Locators in Selenium IDE

Locators in Selenium (Cont.)



- ID - This is the most common technique of locating elements on the web page as ID's are supposed to be unique for each element

```
<html>
<body>
  <form id="loginForm" >
    <input name="username" type="text" />
    <input name="password" type="password" />
    <input name="continue" type="submit" value="Login" />
    <input name="continue" type="button" value="Clear" />
  </form>
</body>
</html>
```

Id = loginForm

3.3: Introduction to Element Locators in Selenium IDE

Locators in Selenium (Cont.)



- Name – Locating elements by their Name is very much similar to locating an element by its ID, except that we use "name=" instead

```
1 <html>
2 <body>
3 <form id="loginForm">
4 <input name="username" type="text" />
5 <input name="password" type="password" />
6 <input name="continue" type="submit" value="Login" />
7 <input name="continue" type="button" value="Clear" />
8 </form>
9 </body>
10 </html>
```

3.3: Introduction to Element Locators in Selenium IDE

Locators in Selenium (Cont.)



- **Link Text** – This type of locator is only used with the hyperlink element. We access the link by prefixing our target with "link=" and then followed by the hyperlink text

```
1 <html>
2 <body>
3   <a href="#">Link1</a>
4   <a href="#">Link2</a>
5   <a href="#">Link3</a>
6   <form id="loginForm">
7     <input name="username" type="text" />
8     <input name="password" type="password" />
9     <input name="continue" type="submit" value="Login" />
10    <input name="continue" type="button" value="Clear" />
11  </form>
12 </body>
13 </html>
```


3.3: Introduction to Element Locators in Selenium IDE



Locating Elements by CSS Selectors

- CSS (Cascading Style Sheets) is a language for describing the rendering of HTML and XML documents
- CSS uses selectors for binding style properties to elements in the document
- Selenium is compatible with CSS 1.0, CSS 2.0, and CSS 3.0 selectors
- CSS Selectors are string patterns used to identify an element based on a combination of HTML tag, id, class, and attributes

3.3: Introduction to Element Locators in Selenium IDE



Locating Elements by CSS Selectors (Cont.)

CSS Selector	Description	Syntax & Example
Tag and ID	tag=HTML Tag id=The ID of the element being accessed	Syntax - css=tag#id Example - css=input#Uname
Tag and Class	tag=HTML Tag class=The class of the element being accessed	Syntax - css=tag.class Example - css=input.inputtext
Tag and Attribute	tag=HTML Tag [attribute=value]	Syntax - css=tag[attribute=value] Example - css=input[name=LName]
Tag, Class and Attribute	tag=HTML Tag class=The class of the element being accessed [attribute=value]	Syntax - css=tag.class[attribute=value] Example - css=input.inputtext[name=LName]]
Inner Text	tag=HTML Tag Inner text=The inner text of the element	Syntax - css=tag:contains("inner text") Example - css=input.contains("Hello")

3.3: Introduction to Element Locators in Selenium IDE

Locating by CSS Selector - Tag and ID



```
<html>
<body>
  <form id="loginForm" >
    <input name="username" type="text" />
    <input name="password" type="password" />
    <input name="continue" type="submit" value="Login" />
    <input name="continue" type="button" value="Clear" />
  </form>
</body>
</html>
```



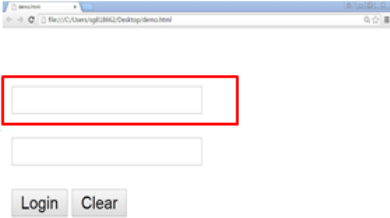
css = form# loginForm

3.3: Introduction to Element Locators in Selenium IDE

Locating by CSS Selector - Tag and Class



```
<html>
<body>
<form id= "loginForm" >
  <input name= "username" type= "text" class= "inputtext" />
  <input name= "password" type= "password" class= "inputtext" />
  <input name= "continue" type= "submit" value= "Login" />
  <input name= "continue" type= "button" value= "Clear" />
</form>
</body>
</html>
```



css = input.inputtext

3.3: Introduction to Element Locators in Selenium IDE

Locating by CSS Selector - Tag and Attribute

The screenshot displays the Selenium IDE interface. On the left, the HTML code for a login form is shown:


```
<html>
<body>
<form id= "loginForm" >
  <input name= "username" type= "text" class= "inputtext" />
  <input name= "password" type= "password" class= "inputtext" />
  <input name= "continue" type= "submit" value= "Login" />
  <input name= "continue" type= "button" value= "Clear" />
</form>
</body>
</html>
```

On the right, the rendered HTML form is shown. The first input field (username) is highlighted with a red box. Below the form, a red box contains the CSS selector:

```
css = input[name=username]
```

3.3: Introduction to Element Locators in Selenium IDE

Locating by CSS Selector – Tag, Class and Attribute



```
<html>
<body>
<form id="loginForm" >
  <input name="username" type="text" class="inputtext" />
  <input name="password" type="password" class="inputtext" />
  <input name="continue" type="submit" value="Login" />
  <input name="continue" type="button" value="Clear" />
</form>
</body>
</html>
```

css = input.inputtext[name=username]

3.3: Introduction to Element Locators in Selenium IDE

Locating by CSS Selector – Inner Text



css=a:contains("Link1")

css=a:contains("Link3")

demo.html

File Edit View Window Help

http://localhost:4444/demo.html

Link1 Link2 Link3

Login Clear

```
1 <html>
2 <body>
3 <a href="#">Link1</a>
4 <a href="#">Link2</a>
5 <a href="#">Link3</a>
6 <form id="loginForm">
7 <input name="username" type="text" />
8 <input name="password" type="password" />
9 <input name="continue" type="submit" value="Login" />
10 <input name="continue" type="button" value="Clear" />
11 </form>
12 </body>
13 </html>
```

3.3: Introduction to Element Locators in Selenium IDE

Locating Elements by DOM



DOM Method	Description	Syntax & Example
getElementById	id of the element: this is the value of the ID attribute of the element to be accessed. This value should always be enclosed in a pair of parentheses ("")	Syntax – document.getElementById("id") Example – document.getElementById("txtName")
getElementsByName	Name: name of the element as defined by its 'name' attribute Index: an integer that indicates which element within getElementsByName's array will be used	Syntax – document.getElementsByName("name")[index] Example – document.getElementsByName("rbGender")[1]
dom:name	Used to locate element contained within a named form.	Syntax – document.forms["name of the form"].elements["name of the element"] Example – document.forms["frmRegister"].elements["usrName"]

3.3: Introduction to Element Locators in Selenium IDE

Locating Elements by DOM (Cont.)



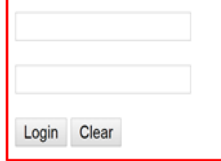
DOM Method	Description	Syntax & Example
dom:index	Used to locate element contained within a form having no name.	Syntax – document.forms[index of the form].elements[index of the element] Example – document.forms[2].elements[3]

3.3: Introduction to Element Locators in Selenium IDE

Locating by DOM – getElementById



```
<html>
<body>
  <form id= "loginForm" >
    <input name= "username" type= "text" class= "inputtext" />
    <input name= "password" type= "password" class= "inputtext" />
    <input name= "continue" type= "submit" value= "Login" />
    <input name= "continue" type= "button" value= "Clear" />
  </form>
</body>
</html>
```



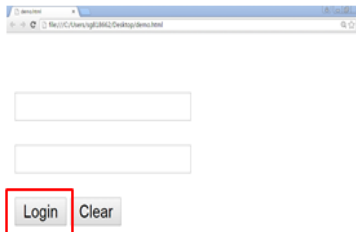
```
document.getElementById(" loginForm ")
```

3.3: Introduction to Element Locators in Selenium IDE

Locating by DOM – getElementByName



```
<html>
<body>
  <form id="loginForm" >
    <input name="username" type="text" class="inputtext" />
    <input name="password" type="password" class="inputtext" />
    <input name="continue" type="submit" value="Login" />
    <input name="continue" type="button" value="Clear" />
  </form>
</body>
</html>
```



```
document.getElementsByName("continue")[1]
```

3.3: Introduction to Element Locators in Selenium IDE

Locating by DOM – dom:name



The screenshot illustrates the process of locating a web element using its DOM path. On the left, the HTML structure of a login form is shown:

```
<html>
<body>
  <form name= "loginForm" >
    <input name= "username" type= "text" class= "inputtext" />
    <input name= "password" type= "password" class= "inputtext" />
    <input name= "continue" type= "submit" value= "Login" />
    <input name= "continue" type= "button" value= "Clear" />
  </form>
</body>
</html>
```

On the right, a visual representation of the login form is shown. The username input field is highlighted with a red rectangle. A line connects this field to the DOM path shown in the code block below:

```
document.forms[" loginForm "].elements[" username "]
```

3.3: Introduction to Element Locators in Selenium IDE

Locating by DOM – dom:index



```
<html>

<body>

<form name= "loginForm" >

<input name= "username" type= "text" class="inputtext" />

→ <input name= "password" type= "password" class="inputtext" />

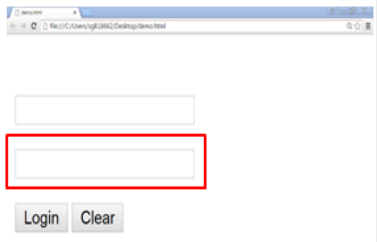
<input name= "continue" type= "submit" value= "Login" />

<input name= "continue" type= "button" value= "Clear" />

</form>

</body>

</html>
```



`document.forms[1].elements[2]`

3.3: Introduction to Element Locators in Selenium IDE



Introduction to XPath

- An XPath can be defined as a query language used for navigating through the XML documents in order to locate different elements
- The basic syntax of an XPath expression is:
 - `//tag[@attributeName='attributeValues']`
 - Difference between single '/' or double '/' - A single slash at the start of Xpath instructs XPath engine to look for element starting from root node. A double slash at the start of Xpath instructs XPath engine to search look for matching element anywhere in the XML document.
 - Tagname – Tagname of a particular element.
 - AttributeName - Attribute name of an element.
 - AttributeValue – Value of an attribute.
- Using the XPath syntax displayed above, we can create multiple XPath expressions

3.3: Introduction to Element Locators in Selenium IDE



Types of XPath

■ Absolute XPath

- The XPath expressions created using absolute XPaths begins the selection from the root node
- These expression either begin with the '/' or the root node and traverse the whole DOM to reach the element
- The advantage of using absolute is, it identifies the element very fast
- Disadvantage here is, if any thing goes wrong or some other tag added in between, then this path will no longer works

■ Relative XPath

- The relative XPath expressions are a lot more compact and use forward double slashes '/'
- These XPaths can select the elements at any location that matches the selection criteria and doesn't necessarily begin with root node
- Advantage of using relative xpath is, you don't need to mention the long xpath, you can start from the middle or in between
- Disadvantage here is, it will take more time in identifying the element as we specify the partial path and not exact path
- If there are multiple elements for the same path, it will select the first element that is identified

3.3: Introduction to Element Locators in Selenium IDE

Locating Elements by XPath



```
<html>

<body>

<form name= "loginForm" >

<input name= "username" type= "text" class="inputtext" />

<input name= "password" type= "password" class="inputtext" />

<input name= "continue" type= "submit" value= "Login" />

<input name= "continue" type= "button" value= "Clear" />

</form>

</body>

</html>
```

//*[*@id*="loginForm"]/input[2]

//html/body/form/input[2]

Demo – UI Element Locators in Selenium IDE Test Case



- Locating UI Elements using various UI Element Locators

3.4: Storing information from the page in the test



Store Commands

- 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
- You can also use Selenium variables to store constants at the beginning of a script
- Selenium variables can be used to store values passed to your test program from the command-line, from another program, or from a file
- 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}`

3.4: Storing information from the page in the test Store Commands (Cont.)



▪ Store

- The plain store command is the most basic of the many store commands and can be used to simply store a constant value in a selenium variable
- It takes two parameters, the text value to be stored and a selenium variable
- Example:

Command	Target	Value
store	Selenium IDE Demo	myVariable
type	name=Textbox1	\${myVariable}

- The above test stores the value "Selenium IDE Demo" in the variable "myVariable"
- You can read the value of the variable in the textbox on your web page named "Textbox1" by setting the value for the type command as \${myVariable}
- Upon execution of above script will store the value "Selenium IDE Demo" in the textbox "Textbox1"

3.4: Storing information from the page in the test Store Commands (Cont.)



▪ storeElementPresent

- This command stores either "true" or "false" depending on the presence of the specified element
- Example:

Command	Target	Value
open		
storeElementPresent	name=loginName	flag1
storeElementPresent	name=Password	flag2

- In the above test script, the variables flag1 & flag2 will store the values either true or false depending upon the presence of the two elements namely "loginName" & "Password"
- We can verify the same by using "echo" command

Command	Target	Value
open		
storeElementPresent	name=loginName	flag1
storeElementPresent	name=Password	flag2
echo	S{flag1}	
echo	S{flag2}	

3.4: Storing information from the page in the test Store Commands (Cont.)



- **storeText**

- This command is used to store the inner text of an element onto a variable
- Example:

Command	Target	Value
open		
storeText	css=h2	textVar
echo	\${textVar}	

- The above script will save the inner text in the variable "textVar" of the element having satisfied the condition i.e. "css=h2"

Demo – Usage of Store Commands in Selenium IDE Test Case



- Using Store Commands

3.5: Working with Alerts



Introduction to Alert Selenium IDE Commands

Command	Description
assertAlert assertNotAlert	Retrieves the message of the alert and asserts it to a string value that you specified.
assertAlertPresent assertAlertNotPresent	Asserts if an Alert is present or not.
storeAlert	Retrieves the alert message and stores it in a variable that you will specify.
storeAlertPresent	Returns TRUE if an alert is present; FALSE if otherwise.
verifyAlert verifyNotAlert	Retrieves the message of the alert and verifies if it is equal to the string value that you specified.
verifyAlertPresent verifyAlertNotPresent	verifies if an Alert is present or not

Introduction to Alert Selenium IDE Commands:

Alerts are probably the simplest form of pop-up windows. The most common Selenium IDE commands used in handling alerts are given in the above slide:

Remember these two things when working with alerts:

1. Selenium IDE will automatically click on the OK button of the alert window and so you will not be able to see the actual alert.
2. Selenium IDE will not be able to handle alerts that are within the page's onload() function. It will only be able to handle alerts that are generated after the page has completely loaded.

Demo – Handling Alert Box in Selenium IDE Test Case



- Working with Alert Selenium IDE Commands

3.5: Working with Confirmation Box



Introduction to Confirmation Selenium IDE Commands

- Confirmations are pop-ups that give you an OK and a CANCEL button, as opposed to alerts which give you only the OK button
- The commands you can use in handling confirmations are similar to those in handling alerts
 - `assertConfirmation/assertNotConfirmation`
 - `assertConfirmationPresent/assertConfirmationNotPresent`
 - `storeConfirmation`
 - `storeConfirmationPresent`
 - `verifyConfirmation/verifyNotConfirmation`
 - `verifyConfirmationPresent/verifyConfirmationNotPresent`
 - `chooseOkOnNextConfirmation/chooseOkOnNextConfirmationAndWait`
 - `chooseCancelOnNextConfirmation`

Demo – Handling Confirmation Dialog Box in Selenium IDE Test Case



- Demo on usage of "chooseOkOnNextConfirmation" and "chooseCancelOnNextConfirmation"

3.6: Debugging in Selenium IDE



Introduction to Debugging in Selenium IDE

- Debugging means finding and fixing errors in your test case
- This is a normal part of test case development
- Sometimes, as a test automator, you will need to debug your tests to see what is wrong
- There are various commonly used techniques available in Selenium IDE which can be used to identify an error in the test case
- The tester can optionally break or start the execution of a test case to debug and figure out the existing error in the test case

3.6: Debugging in Selenium IDE



Using Breakpoints in Test Case

- One can run up to a specific command in the middle of the test case and inspect how the test case behaves at that point
- To do this, set a breakpoint on the command just before the one to be examined
- Steps to be followed
 - Select a command
 - Right-click, and from the context menu select Toggle Breakpoint
 - Then click the Run button to run your test case from the beginning up to the breakpoint
 - Click on Step button to execute the test case which has halted as it has reached the breakpoint
 - Observer the test execution

3.6: Debugging in Selenium IDE

Using Startpoint in Test Case

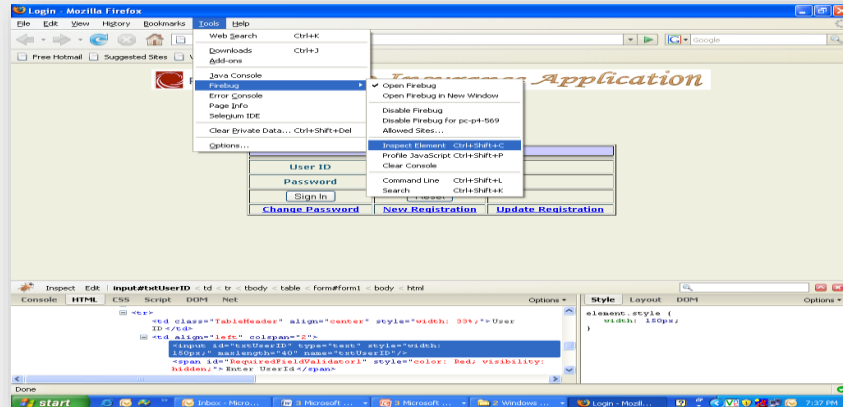


- If you have a really long test and it's failing towards the end, then you can set a custom start point so that you don't have to run the entire test when you're investigating the failure
- For example, your test might register a new user, log in, and then fail on the Home Page
- You could simply navigate to the home page yourself and set your test to start from there
- To set a start point simply right click on the first command you want Selenium IDE to execute and click 'Set / Clear Start Point'
- You will see a small play icon appear to the left of your command

3.7: Object Identification in Selenium IDE Using Firebug to identify object



- Firebug is add on to Firefox
- It helps in getting object properties, DOM structure, CSS details and XPath



3.7: Object Identification in Selenium IDE

Create Script Using Selenium IDE

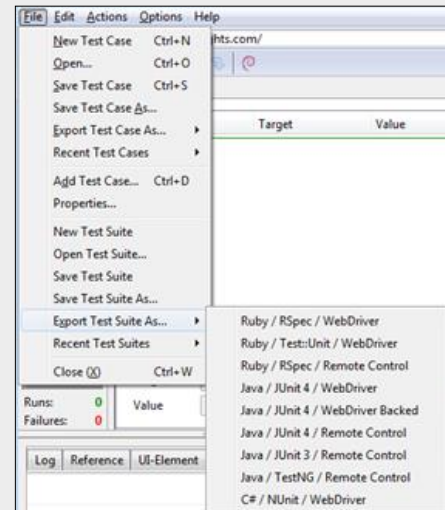


- Perform following steps to create script:
 - Launch Mozilla Firefox
 - Open application in Firefox
 - Invoke Selenium Tools -> Selenium IDE
 - Invoke firebug Tools -> firebug -> Open Firebug
 - Enter command in Selenium IDE
 - Inspect element using firebug and specify element locator
 - Specify value if required
 - Repeat above steps as required

3.8: Export Scripts to Multiple Languages

Exporting scripts to multiple languages and Formats

- Test cases can be exported only to the following formats:
 - .cs (C# source code)
 - .java (Java source code)
 - .py (Python source code)
 - .rb (Ruby source code)



Summary



In this lesson, you have learnt

- Selenium IDE (Integrated Development Environment) is the simplest tool in the Selenium Suite.
- Menu bar is used in creating, modifying, and exporting test cases into formats useable by Selenium RC and WebDriver
- The default format for Selenese commands is HTML.
- The Test Case Pane shows the list of currently opened test cases and a concise summary of test runs
- Locators tell Selenium IDE which GUI elements (say Text Box, Buttons, Check Boxes etc.) its needs to operate on
- The choice of locator depends largely on your Application Under Test



Add the notes here.

Review Question



Question 1

- Select the Locator which is NOT part of Selenium IDE
- CSS Selector
- XPath
- getElementById
- getElementByXpath



Question 2: True/False

- The choice of locator depends largely on your Application Under Test.

Question 3: Fill in the Blanks

- Error messages and information messages showing the progress are displayed automatically in _____ pane.