**How to use Selenium IDE with Scripts & Commands (Assertions, Actions)**

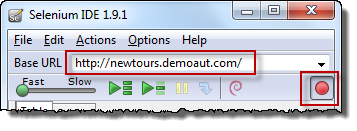
We will use the Mercury Tours website as our web application under test. It is an online flight reservation system that contains all the elements we need for this tutorial. Its URL is <http://demo.guru99.com/test/newtours/>, and this will be our Base URL.

**Create a Script by Recording**

Let us now create our first test script in Selenium IDE using the most common method - by recording. Afterward, we shall execute our script using the playback feature.

**Step 1**

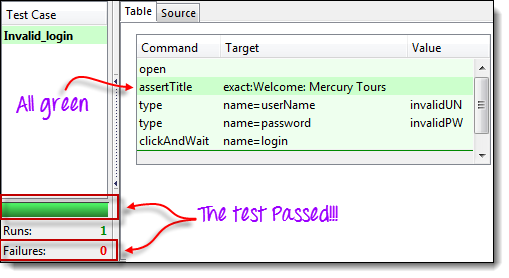
* Launch Firefox and Selenium IDE.
* Type the value for our Base URL: <http://demo.guru99.com/test/newtours/>.
* Toggle the Record button on (if it is not yet toggled on by default).

[](https://www.guru99.com/images/step_1(1).png)

|  |  |
| --- | --- |
| **Step 2**  In Firefox, navigate to <http://demo.guru99.com/test/newtours/>. Firefox should take you to the page similar to the one shown below.  [How to use Selenium IDE with Scripts & Commands (Assertions, Actions)](https://www.guru99.com/images/step_2(1).png) | |
| **Step 3**   * Right-click on any blank space within the page, like on the Mercury Tours logo on the upper left corner. This will bring up the Selenium IDE context menu. Note: Do not click on any hyperlinked objects or images * Select the "Show Available Commands" option. * Then, select "assertTitle exact: Welcome: Mercury Tours." This is a command that makes sure that the page title is correct. | |
| [How to use Selenium IDE with Scripts & Commands (Assertions, Actions)](https://www.guru99.com/images/step_3(1).png)  [How to use Selenium IDE with Scripts & Commands (Assertions, Actions)](https://www.guru99.com/images/editor_-_after_assert_title_-_before_logging_in.png) | |
| **Step 4**   * In the "User Name" text box of Mercury Tours, type an invalid username, "invalidUNN". * In the "Password" text box, type an invalid password, "invalidPWD". | |
| [How to use Selenium IDE with Scripts & Commands (Assertions, Actions)](https://www.guru99.com/images/Step4a.png) | [How to use Selenium IDE with Scripts & Commands (Assertions, Actions)](https://www.guru99.com/images/step4b.png) |
| **Step 5**   * Click on the "Sign-In" button. Firefox should take you to this page.   [How to use Selenium IDE with Scripts & Commands (Assertions, Actions)](https://www.guru99.com/images/step_5a.png) | |
| **Step 6**  Toggle the record button off to stop recording. Your script should now look like the one shown below.  [How to use Selenium IDE with Scripts & Commands (Assertions, Actions)](https://www.guru99.com/images/step_6.png) | |
| **Step 7**  Now that we are done with our test script, we shall save it in a test case. In the File menu, select "Save Test Case". Alternatively, you can simply press Ctrl+S.  [How to use Selenium IDE with Scripts & Commands (Assertions, Actions)](https://www.guru99.com/images/step_7.png) | |
| **Step 8**   * Choose your desired location, and then name the [Test Case](https://www.guru99.com/test-case.html) as "Invalid\_login". * Click the "Save" button.   [How to use Selenium IDE with Scripts & Commands (Assertions, Actions)](https://www.guru99.com/images/step_8.png) | |
| **Step 9.**  Notice that the file was saved as HTML.  [How to use Selenium IDE with Scripts & Commands (Assertions, Actions)](https://www.guru99.com/images/step_9.png) | |

**Step 10.**

Go back to Selenium IDE and click the Playback button to execute the whole script. Selenium IDE should be able to replicate everything flawlessly.

[](https://www.guru99.com/images/step_10.png)

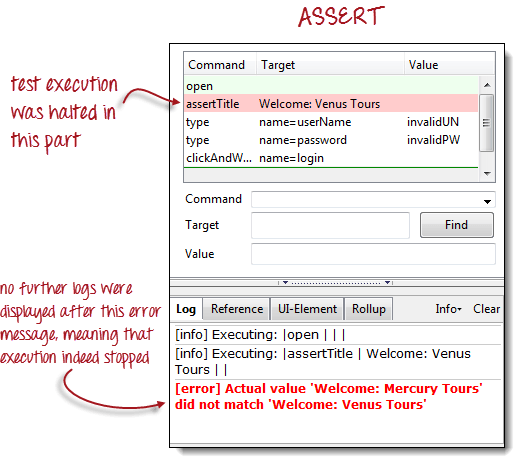
**Introduction to Selenium Commands - Selenese**

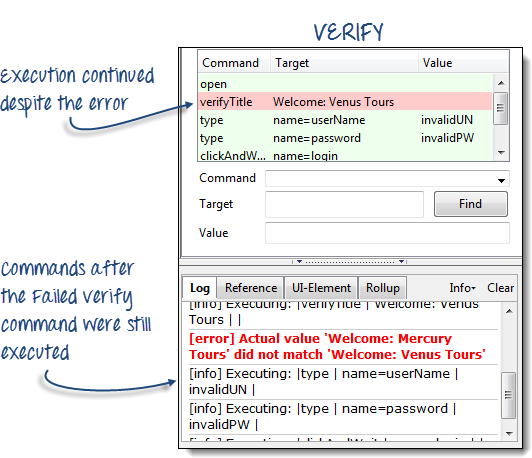
* Selenese commands can have up to a maximum of two parameters: target and value.
* Parameters are not required all the time. It depends on how many the command will need.

**3 Types of Commands**

|  |  |
| --- | --- |
| **Actions** | These are commands that directly interact with page elements.  Example: the "click" command is an action because you directly interact with the element you are clicking at.  The "type" command is also an action because you are putting values into a text box, and the text box shows them to you in return. There is a two-way interaction between you and the text box. |
| **Accessors** | They are commands that allow you to store values to a variable.  Example: the "storeTitle" command is an accessor because it only "reads" the page title and saves it in a variable. It does not interact with any element on the page. |
| **Assertions** | They are commands that verify if a certain condition is met.  **3 Types of Assertions**   * **Assert**. When an "assert" command fails, the test is stopped immediately. * **Verify**. When a "verify" command fails, Selenium IDE logs this failure and continues with the test execution. * **WaitFor**. Before proceeding to the next command, "waitFor" commands will first wait for a certain condition to become true.   + If the condition becomes true within the waiting period, the step passes.   + If the condition does not become true, the step fails. Failure is logged, and test execution proceeds to the next command.   + By default, the timeout value is set to 30 seconds. You can change this in the Selenium IDE Options dialog under the General tab. |

**Assert vs. Verify**

[](https://www.guru99.com/images/assert.png)

[](https://www.guru99.com/images/verify.png)

**Common Commands**

|  |  |  |
| --- | --- | --- |
| **Command** | **Number of Parameters** | **Description** |
| open | 0 - 2 | Opens a page using a URL. |
| click/clickAndWait | 1 | Clicks on a specified element. |
| type/typeKeys | 2 | Types a sequence of characters. |
| verifyTitle/assertTitle | 1 | Compares the actual page title with an expected value. |
| verifyTextPresent | 1 | Checks if a certain text is found within the page. |
| verifyElementPresent | 1 | Checks the presence of a certain element. |
| verifyTable | 2 | Compares the contents of a table with expected values. |
| waitForPageToLoad | 1 | Pauses execution until the page is loaded completely. |
| waitForElementPresent | 1 | Pauses execution until the specified element becomes present. |

**Create a Script Manually with Firebug**

Now, we shall recreate the same test case manually, by typing in the commands. This time, we will need to use Firebug.

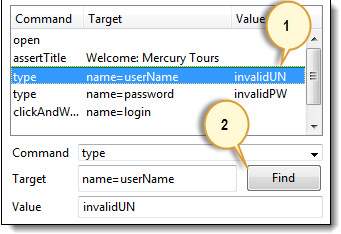
|  |
| --- |
| **Step 1**   * Open Firefox and Selenium IDE. * Type the base URL (<http://demo.guru99.com/test/newtours/>). * The record button should be OFF.   [How to use Selenium IDE with Scripts & Commands (Assertions, Actions)](https://www.guru99.com/images/ToggleOff.png) |
| **Step 2:** Click on the topmost blank line in the Editor.  [How to use Selenium IDE with Scripts & Commands (Assertions, Actions)](https://www.guru99.com/images/editor_-_step_2a.png) |
| Type "open" in the Command text box and press Enter.  [How to use Selenium IDE with Scripts & Commands (Assertions, Actions)](https://www.guru99.com/images/ide_-_step_2.png) |
| **Step 3**   * Navigate Firefox to our base URL and activate Firebug * In the Selenium IDE Editor pane, select the second line (the line below the "open" command) and create the second command by typing "assertTitle" on the Command box. * Feel free to use the autocomplete feature.   [How to use Selenium IDE with Scripts & Commands (Assertions, Actions)](https://www.guru99.com/images/autocomplete(1).png) |
| **Step 4**   * In Firebug, expand the <head> tag to display the <title> tag. * Click on the value of the <title> tag (which is "Welcome: Mercury Tours") and paste it onto the Target field in the Editor.   [How to use Selenium IDE with Scripts & Commands (Assertions, Actions)](https://www.guru99.com/images/Step4manual.png) |
| **Step 5**   * To create the third command, click on the third blank line in the Editor and key-in "type" on the Command text box. * In Firebug, click on the "Inspect" button.   [How to use Selenium IDE with Scripts & Commands (Assertions, Actions)](https://www.guru99.com/images/Step5mnual.png) |
| Click on the User Name text box. Notice that Firebug automatically shows you the HTML code for that element.  [How to use Selenium IDE with Scripts & Commands (Assertions, Actions)](https://www.guru99.com/images/ste5manual1.png) |
| **Step 6**  Notice that the User Name text box does not have an ID, but it has a NAME attribute. We shall, therefore, use its NAME as the locator. Copy the NAME value and paste it onto the Target field in Selenium IDE.  [How to use Selenium IDE with Scripts & Commands (Assertions, Actions)](https://www.guru99.com/images/step6a.png)  Still in the Target text box, prefix "userName" with "name=", indicating that Selenium IDE should target an element whose NAME attribute is "userName."  [How to use Selenium IDE with Scripts & Commands (Assertions, Actions)](https://www.guru99.com/images/step6b.png)  Type "invalidUN" in the Value text box of Selenium IDE. Your test script should now look like the image below. We are done with the third command. Note: Instead of invalidUN, you may enter any other text string. But Selenium IDE is case sensitive, and you type values/attributes exactly like in the application.  [How to use Selenium IDE with Scripts & Commands (Assertions, Actions)](https://www.guru99.com/images/Step6c.png) |
| **Step 7**   * To create the fourth command, key-in "type" on the Command text box. * Again, use Firebug's "Inspect" button to get the locator for the "Password" text box.   [How to use Selenium IDE with Scripts & Commands (Assertions, Actions)](https://www.guru99.com/images/firebug_-_step_8.png)   * Paste the NAME attribute ("password") onto the Target field and prefix it with "name=" * Type "invalidPW" in the Value field in Selenium IDE. Your test script should now look like the image below.   [How to use Selenium IDE with Scripts & Commands (Assertions, Actions)](https://www.guru99.com/images/ide_-_step_8.png) |
| **Step 8**   * For the fifth command, type "clickAndWait" on the Command text box in Selenium IDE. * Use Firebug's "Inspect" button to get the locator for the "Sign In" button.   [How to use Selenium IDE with Scripts & Commands (Assertions, Actions)](https://www.guru99.com/images/firebug_-_step_9.png)   * Paste the value of the NAME attribute ("login") onto the Target text box and prefix it with "name=". * Your test script should now look like the image below.   [How to use Selenium IDE with Scripts & Commands (Assertions, Actions)](https://www.guru99.com/images/Step8a.png) |

**Step 9:** Save the test case in the same way as we did in the previous section.

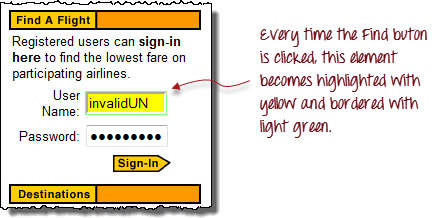
**Using the Find Button**

**The Find button in Selenium IDE is used to verify if what we had put in the Target text box is indeed the correct UI element.**

Let us use the Invalid\_login test case that we created in the previous sections. Click on any command with a Target entry, say, the third command.

[](https://www.guru99.com/images/find(1).png)

Click on the Find button. Notice that the User Name text box within the Mercury Tours page becomes highlighted for a second.

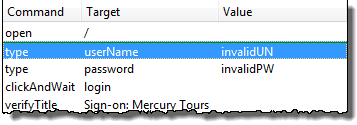
[](https://www.guru99.com/images/firefox_-_user_name_highlighted(1).png)

This indicates that Selenium IDE was able to detect and access the expected element correctly. If the Find button highlighted a different element or no element at all, then there must be something wrong with your script.

**Execute Command**

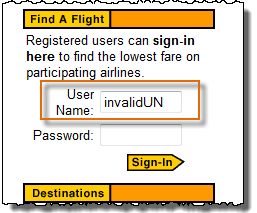
**This allows you to execute any single command without running the whole test case**. Just click on the line you wish to execute and then either click on "Actions > Execute this command" from the menu bar or simply press "X" on your keyboard.

**Step 1.** Make sure that your browser is on the Mercury Tours homepage. Click on the command you wish to execute. In this example, click on the "type | userName | invalidUN" line.

[](https://www.guru99.com/images/execute1.png)

**Step 2.** Press "X" on your keyboard.

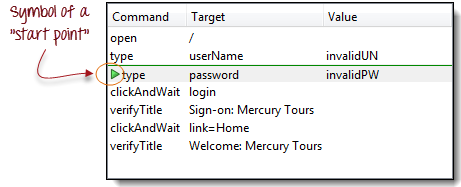
**Step 3.** Observe that the text box for username becomes populated with the text "invalidUN"

[](https://www.guru99.com/images/execute2.png)

**Executing commands this way is highly dependent on the page that Firefox is currently displaying**. This means that if you try the example above with the Google homepage displayed instead of Mercury Tours', then your step will fail because there is no text box with a "userName" attribute within Google's homepage.

**Start point**

**A start point is an indicator that tells Selenium IDE which lines the execution will start**. **Its shortcut key is "S".**

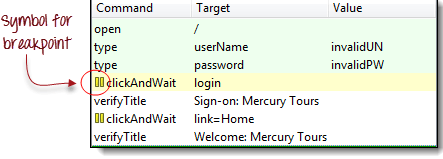
[](https://www.guru99.com/images/startpoint.png)

In the example above, playback will start on the third line (type | password | invalidPW). **You can only have one start point in a single test script.**

The start point is similar to Execute Command in such that they are dependent on the currently displayed page. The start point will fail if you are on the wrong page.

**Breakpoints**

Breakpoints are indicators that tell Selenium IDE where to automatically pause the test. **The shortcut key is "B".**

[](https://www.guru99.com/images/BreakPoint.png)

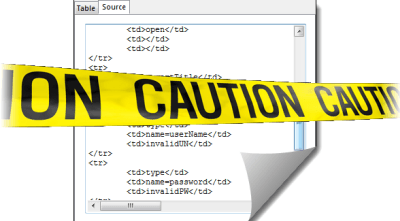
The yellow highlight means that the current step is pending. This proves that Selenium IDE has paused execution on that step. **You can have multiple breakpoints in one test case.**

**Step**

It allows you to execute succeeding commands one at a time after pausing the test case. Let us use the scenario in the previous section "Breakpoints."

|  |  |
| --- | --- |
| [How to use Selenium IDE with Scripts & Commands (Assertions, Actions)](https://www.guru99.com/images/Stepa.png) | **Before clicking "Step."**  The test case pauses at the line "clickAndWait | login". |
| [How to use Selenium IDE with Scripts & Commands (Assertions, Actions)](https://www.guru99.com/images/Stepb.png) | **After clicking "Step."**  The "clickAndWait | login" line is run and pauses to the next command (verifyTitle | Sign-on: Mercury Tours).  Notice that the next line is paused even though there is no breakpoint there. This is the main purpose of the Step feature - it executes the succeeding commands one at a time to give you more time to inspect the outcome after each step. |

**Important Things to Note When Using Other Formats in Source View**

[](https://www.guru99.com/images/Caution.png)

**Selenium IDE works well only with HTML - other formats are still in experimental mode**. It is **NOT advisable** to create or edit tests using other formats in Source View because there is still a lot of work needed to make it stable. Below are the known bugs as of version 1.9.1.

* You will not be able to perform playback nor switch back to Table View unless you revert to HTML.
* The only way to add commands safely on the source code is by recording them.
* When you modify the source code manually, all of it will be lost when you switch to another format.
* Though you can save your test case while in Source View, Selenium IDE will not be able to open it.

**How to use Locators in Selenium IDE**

**What are Locators?**

Locator is a command that tells Selenium IDE which GUI elements ( say Text Box, Buttons, Check Boxes etc) its needs to operate on.  Identification of correct GUI elements is a prerequisite to creating an automation script. But accurate identification of GUI elements is more difficult than it sounds. Sometimes, you end up working with incorrect GUI elements or no elements at all!  Hence, Selenium provides a number of Locators to precisely locate a GUI element

The different types of Locators in Selenium IDE

* 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

There are commands that do not need a locator (such as the "open" command). However, most of them do need Locators.

**The choice of locator depends largely on your Application Under Test**. In this tutorial, we will toggle between Facebook, new tours.demoaut on the basis of locators that these applications support. Likewise in your [Testing](https://www.guru99.com/software-testing.html) project, you will select any of the above-listed locators based on your application support.

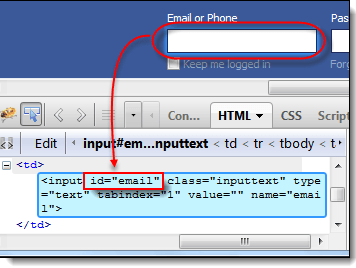
**Locating by ID**

This is the most common way of locating elements since ID's are supposed to be unique for each element.

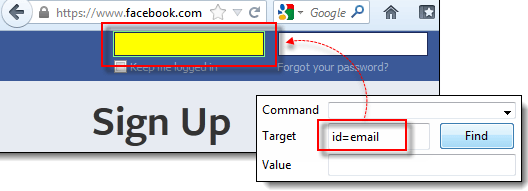
**Target Format:** id=*id of the element*

For this example, we will use Facebook as our test app because Mercury Tours do not use ID attributes.

**Step 1.** Since this tutorial was created, Facebook has changed their Login Page Design. Use this demo page <http://demo.guru99.com/test/facebook.html> for testing. Inspect the "Email or Phone" text box using Firebug and take note of its ID. In this case, the ID is "email."

[](https://www.guru99.com/images/locator1.png)

**Step 2.** Launch Selenium IDE and enter "id=email" in the Target box. Click the Find button and notice that the "Email or Phone" text box becomes highlighted with yellow and bordered with green, meaning, Selenium IDE was able to locate that element correctly.

[](https://www.guru99.com/images/locator2.png)

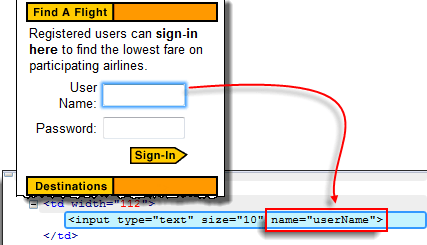
**Locating by Name**

Locating elements by name are very similar to locating by ID, except that we use the **"name="** prefix instead.

**Target Format:** name=*name of the element*

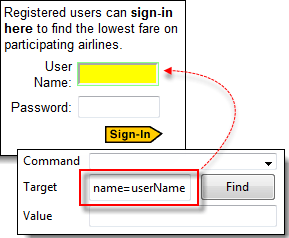
In the following demonstration, we will now use Mercury Tours because all significant elements have names.

**Step 1.** Navigate to <http://demo.guru99.com/test/newtours/> and use Firebug to inspect the "User Name" text box. Take note of its name attribute.

[](https://www.guru99.com/images/locator3.png)

Here, we see that the element's name is "userName".

**Step 2.** In Selenium IDE, enter "name=userName" in the Target box and click the Find button. Selenium IDE should be able to locate the User Name text box by highlighting it.

[](https://www.guru99.com/images/locator4.png)

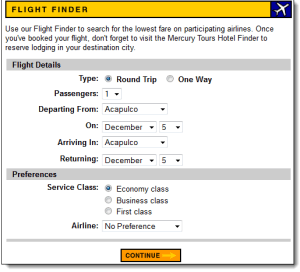
**Locating by Name using Filters**

Filters can be used when multiple elements have the same name. **Filters are additional attributes used to distinguish elements with the same name.**

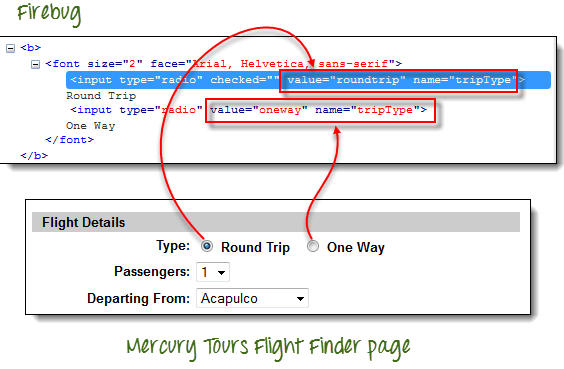
**Target Format**: name=*name\_of\_the\_element* *filter*=*value\_of\_filter*

Let's see an example -

**Step 1**. Log on to Mercury Tours using "tutorial" as the username and password. It should take you to the Flight Finder page shown below.

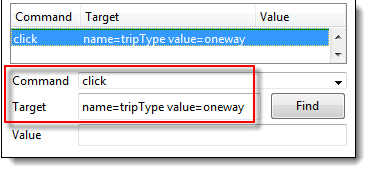
[](https://www.guru99.com/images/locator5.png)

**Step 2.** Using Firebug, notice that the Round Trip and One Way radio buttons have the same name "tripType." However, they have different VALUE attributes so we can use each of them as our filter.

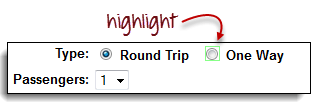
[](https://www.guru99.com/images/locator6.png)

**Step 3.**

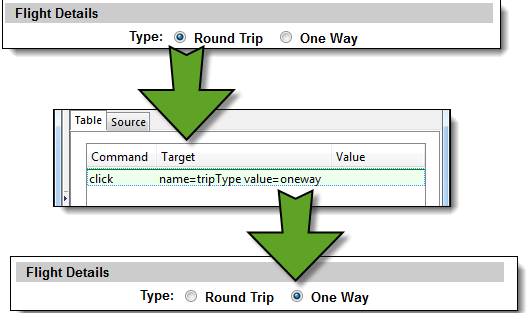
* We are going to access the One Way radio button first. Click the first line on the Editor.
* In the Command box of Selenium IDE, enter the command "click".
* In the Target box, enter "name=tripType value=oneway".  The "value=oneway" portion is our filter.

[](https://www.guru99.com/images/locator7.png)

**Step 4**. Click the Find button and notice that Selenium IDE is able to highlight the One Way radio button with green - meaning that we are able to access the element successfully using its VALUE attribute.

[](https://www.guru99.com/images/locator8.png)

**Step 5.** Press the "X" key in your keyboard to execute this click command. Notice that the One Way radio button became selected.

[](https://www.guru99.com/images/locator9.png)

You can do the exact same thing with the Round Trip radio button, this time, using "name=tripType value=roundtrip" as your target.

**Locating by Link Text**

This type of locator applies only to hyperlink texts. We access the link by prefixing our target with "link=" and then followed by the hyperlink text.

**Target Format**: link=*link\_text*

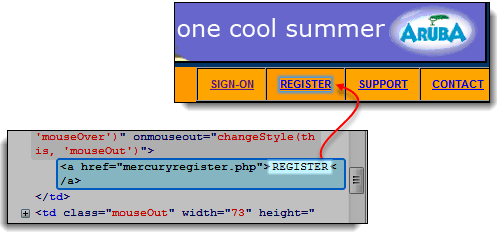
In this example, we shall access the "REGISTER" link found on the Mercury Tours homepage.

**Step 1.**

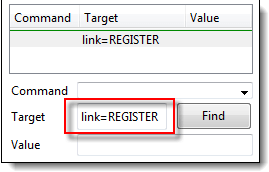
* First, make sure that you are logged off from Mercury Tours.
* Go to Mercury Tours homepage.

**Step 2**.

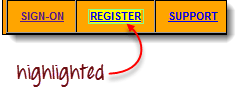
* Using Firebug, inspect the "REGISTER" link. The link text is found between and tags.
* In this case, our link text is "REGISTER". Copy the link text.

[](https://www.guru99.com/images/locator10.png)

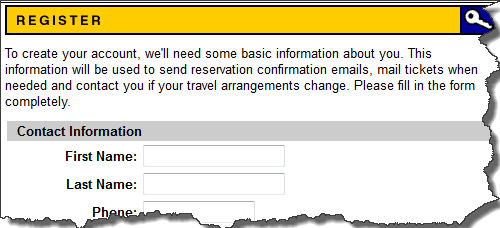
**Step 3**. Copy the link text in Firebug and paste it onto Selenium IDE's Target box. Prefix it with "link=".

[](https://www.guru99.com/images/locator11.png)

**Step 4.** Click on the Find button and notice that Selenium IDE was able to highlight the REGISTER link correctly.

[](https://www.guru99.com/images/locator12.png)

**Step 5.** To verify further, enter "clickAndWait" in the Command box and execute it. Selenium IDE should be able to click on that REGISTER link successfully and take you to the Registration page shown below.

[](https://www.guru99.com/images/locator13.png)

**Locating by CSS Selector**

**CSS Selectors are string patterns used to identify an element based on a combination of HTML tag, id, class, and attributes**. **Locating by CSS Selector is more complicated than the previous methods, but it is the most common locating strategy of advanced Selenium users because it can access even those elements that have no ID or name.**

CSS Selectors have many formats, but we will only focus on the most common ones.

* Tag and ID
* Tag and class
* Tag and attribute
* Tag, class, and attribute
* Inner text

When using this strategy, we always prefix the Target box with "css=" as will be shown in the following examples.

**Locating by CSS Selector - Tag and ID**

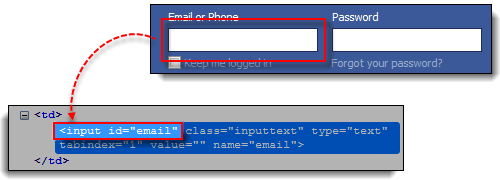
Again, we will use Facebook's Email text box in this example. As you can remember, it has an ID of "email," and we have already accessed it in the "Locating by ID" section. This time, we will use a CSS Selector with ID in accessing that very same element.

|  |  |
| --- | --- |
| **Syntax** | **Description** |
| css=*tag*#*id* | * tag = the HTML tag of the element being accessed * # = the hash sign. This should always be present when using a CSS Selector with ID * id = the ID of the element being accessed |

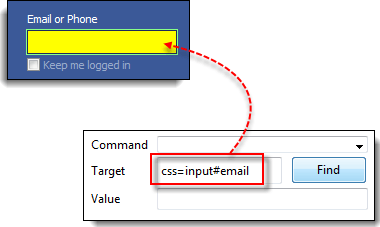
**Keep in mind that the ID is always preceded by a hash sign (#).**

**Step 1.** Navigate to [www.facebook.com](http://www.facebook.com/). Using Firebug, examine the "Email or Phone" text box.

At this point, take note that the HTML tag is "input" and its ID is "email". So our syntax will be "css=input#email".

[](https://www.guru99.com/images/locator14.png)

**Step 2.** Enter "css=input#email" into the Target box of Selenium IDE and click the Find button. Selenium IDE should be able to highlight that element.

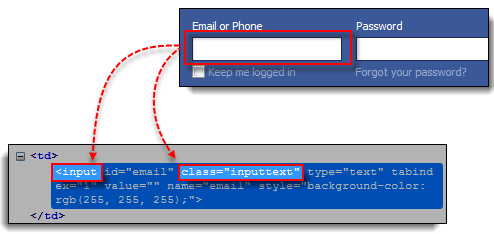
[](https://www.guru99.com/images/locator15.png)

**Locating by CSS Selector - Tag and Class**

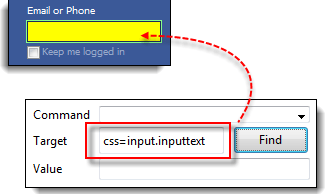
Locating by CSS Selector using an HTML tag and a class name is similar to using a tag and ID, but in this case, a dot (.) is used instead of a hash sign.

|  |  |
| --- | --- |
| **Syntax** | **Description** |
| css=*tag*.*class* | * tag = the HTML tag of the element being accessed * . = the dot sign. This should always be present when using a CSS Selector with class * class = the class of the element being accessed |

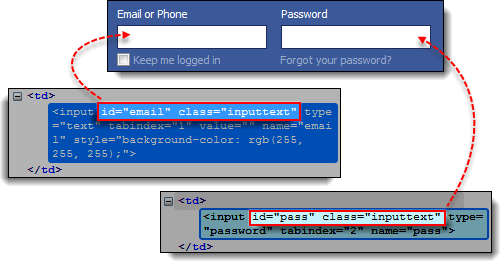
**Step 1.** Go to the demo page <http://demo.guru99.com/test/facebook.html> and use Firebug to inspect the "Email or Phone" text box. Notice that its HTML tag is "input" and its class is "inputtext."

[](https://www.guru99.com/images/locator16.png)

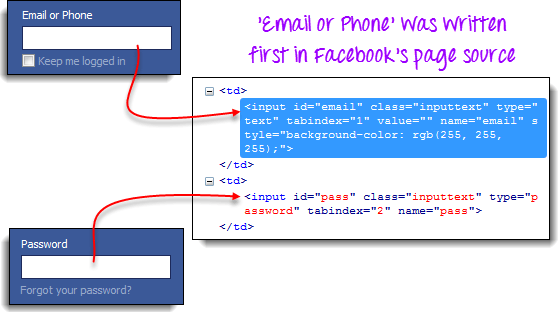
**Step 2.** In Selenium IDE, enter "css=input.inputtext" in the Target box and click Find. Selenium IDE should be able to recognize the Email or Phone text box.

[](https://www.guru99.com/images/locator17.png)

**Take note that when multiple elements have the same HTML tag and name, only the first element in source code will be recognized**. Using Firebug, inspect the Password text box in Facebook and notice that it has the same name as the Email or Phone text box.

[](https://www.guru99.com/images/locator18.png)

The reason why only the Email or Phone text box was highlighted in the previous illustration is that it comes first in Facebook's page source.

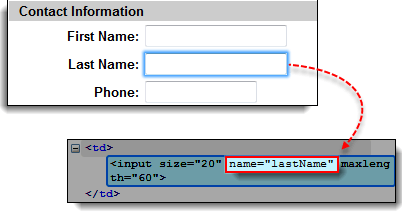
[](https://www.guru99.com/images/locator19.png)

**Locating by CSS Selector - Tag and Attribute**

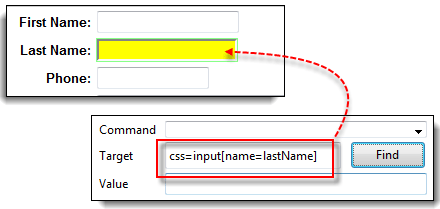
This strategy uses the HTML tag and a specific attribute of the element to be accessed.

|  |  |
| --- | --- |
| **Syntax** | **Description** |
| css=*tag*[*attribute*=*value*] | * tag = the HTML tag of the element being accessed * [ and ] = square brackets within which a specific attribute and its corresponding value will be placed * attribute = the attribute to be used. It is advisable to use an attribute that is unique to the element such as a name or ID. * value = the corresponding value of the chosen attribute. |

**Step 1.** Navigate to Mercury Tours' Registration page (<http://demo.guru99.com/test/newtours/register.php>) and inspect the "Last Name" text box. Take note of its HTML tag ("input" in this case) and its name ("lastName").

[](https://www.guru99.com/images/locator20.png)

**Step 2.** In Selenium IDE, enter "css=input[name=lastName]" in the Target box and click Find. Selenium IDE should be able to access the Last Name box successfully.

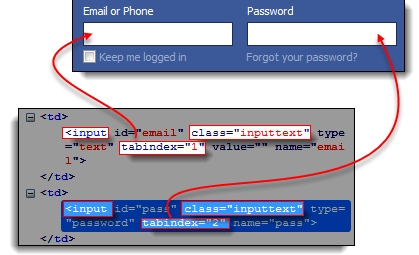
[](https://www.guru99.com/images/locator21.png)

**When multiple elements have the same HTML tag and attribute, only the first one will be recognized**. This behavior is similar to locating elements using CSS selectors with the same tag and class.

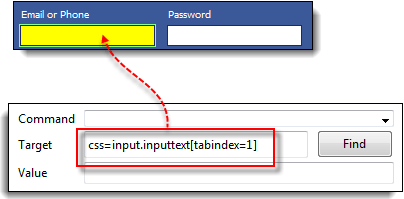
**Locating by CSS Selector - tag, class, and attribute**

|  |  |
| --- | --- |
| **Syntax** | **Description** |
| css=*tag.class*[*attribute*=*value*] | * tag = the HTML tag of the element being accessed * . = the dot sign. This should always be present when using a CSS Selector with class * class = the class of the element being accessed * [ and ] = square brackets within which a specific attribute and its corresponding value will be placed * attribute = the attribute to be used. It is advisable to use an attribute that is unique to the element such as a name or ID. * value = the corresponding value of the chosen attribute. |

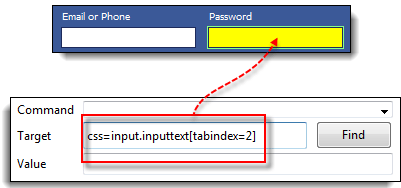
**Step 1.** Go to the demo page <http://demo.guru99.com/test/facebook.html> and use Firebug to inspect the 'Email or Phone' and 'Password' input boxes. Take note of their HTML tag, class, and attributes. For this example, we will select their 'tabindex' attributes.

[](https://www.guru99.com/images/locator22.png)

**Step 2.**  We will access the 'Email or Phone' text box first. Thus, we will use a tabindex value of 1. Enter "css=input.inputtext[tabindex=1]" in Selenium IDE's Target box and click Find. The 'Email or Phone' input box should be highlighted.

[](https://www.guru99.com/images/locator23.png)

**Step 3.** To access the Password input box, simply replace the value of the tabindex attribute. Enter "css=input.inputtext[tabindex=2]" in the Target box and click on the Find button. Selenium IDE must be able to identify the Password text box successfully.

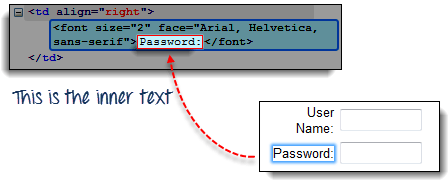
[](https://www.guru99.com/images/locator24.png)

**Locating by CSS Selector - inner text**

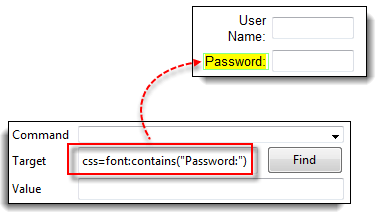
As you may have noticed, HTML labels are seldom given id, name, or class attributes. So, how do we access them? The answer is through the use of their inner texts. **Inner texts are the actual string patterns that the HTML label shows on the page.**

|  |  |
| --- | --- |
| **Syntax** | **Description** |
| css=*tag*:contains("*inner text*") | * tag = the HTML tag of the element being accessed * inner text = the inner text of the element |

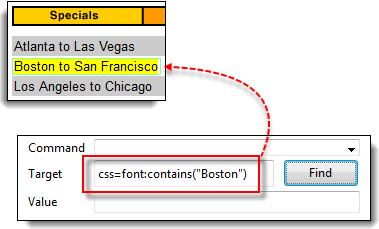
**Step 1.** Navigate to Mercury Tours' homepage (<http://demo.guru99.com/test/newtours/>) and use Firebug to investigate the "Password" label. Take note of its HTML tag (which is "font" in this case) and notice that it has no class, id, or name attributes.

[](https://www.guru99.com/images/locator25.png)

**Step 2.** Type **css=font:contains("Password:")** into Selenium IDE's Target box and click Find. Selenium IDE should be able to access the Password label as shown in the image below.

[](https://www.guru99.com/images/locator26.png)

**Step 3.** This time, replace the inner text with "Boston" so that your Target will now become "css=font:contains("Boston")". Click Find. You should notice that the "Boston to San Francisco" label becomes highlighted. This shows you that Selenium IDE can access a long label even if you just indicated the first word of its inner text.

[](https://www.guru99.com/images/locator27.png)

**Locating by DOM (Document Object Model)**

The Document Object Model (DOM), in simple terms, is the way by which HTML elements are structured. Selenium IDE is able to use the DOM in accessing page elements. If we use this method, our Target box will always start with "dom=document..."; however, the "dom=" prefix is normally removed because Selenium IDE is able to automatically interpret anything that starts with the keyword "document" to be a path within the DOM anyway.

There are four basic ways to locate an element through DOM:

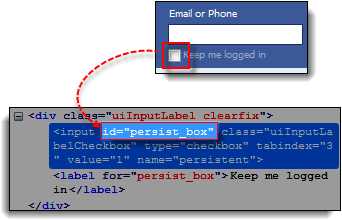
* getElementById
* getElementsByName
* dom:name (applies only to elements within a named form)
* dom:index

Locating by DOM - getElementById

Let us focus on the first method - using the getElementById method. The syntax would be:

|  |  |
| --- | --- |
| **Syntax** | **Description** |
| document.getElementById("*id of the element*") | 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 (""). |

**Step 1.** Use this demo page <http://demo.guru99.com/test/facebook.html> Navigate to it and use Firebug to inspect the "Keep me logged in" check box. Take note of its ID.

[](https://www.guru99.com/images/locator28.png)

We can see that the ID we should use is "persist\_box".

**Step 2.** Open Selenium IDE and in the Target box, enter "document.getElementById("persist\_box")" and click Find. Selenium IDE should be able to locate the "Keep me logged in" check box. Though it cannot highlight the interior of the check box, Selenium IDE can still surround the element with a bright green border as shown below.

[](https://www.guru99.com/images/locator29.png)

**Locating by DOM - getElementsByName**

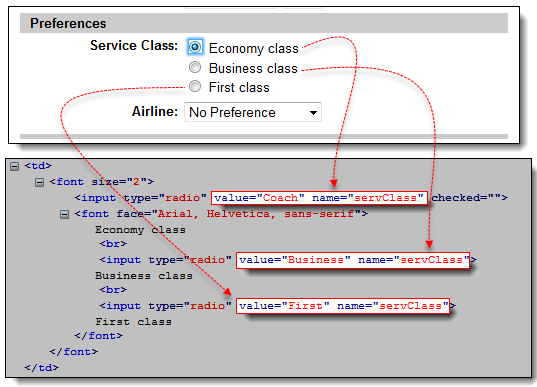
The getElementById method can access only one element at a time, and that is the element with the ID that you specified. The getElementsByName method is different. It collects an array of elements that have the name that you specified. You access the individual elements using an index which starts at 0.

|  |  |  |
| --- | --- | --- |
| [How to use Locators in Selenium IDE](https://www.guru99.com/images/locator30.png) | **getElementById**   * It will get only one element for you. * That element bears the ID that you specified inside the parentheses of getElementById(). | |
| [How to use Locators in Selenium IDE](https://www.guru99.com/images/locator31.jpg) | | **getElementsByName**   * It will get a collection of elements whose names are all the same. * Each element is indexed with a number starting from 0 just like an array * You specify which element you wish to access by putting its index number into the square brackets in getElementsByName's syntax below. |

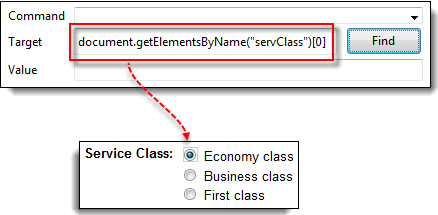
|  |  |
| --- | --- |
| **Syntax** | **Description** |
| document.getElementsByName("*name*")[*index*] | * name = name of the element as defined by its 'name' attribute * index = an integer that indicates which element within getElementsByName's array will be used. |

**Step 1.** Navigate to Mercury Tours' Homepage and login using "tutorial" as the username and password. Firefox should take you to the Flight Finder screen.

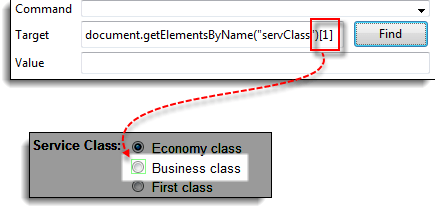
**Step 2.** Using Firebug, inspect the three radio buttons at the bottom portion of the page (Economy class, Business class, and First class radio buttons). Notice that they all have the same name which is "servClass".

[](https://www.guru99.com/images/locator32.png)

**Step 3.** Let us access the "Economy class" radio button first. Of all these three radio buttons, this element comes first, so it has an index of 0. In Selenium IDE, type "document.getElementsByName("servClass")[0]" and click the Find button. Selenium IDE should be able to identify the Economy class radio button correctly.

[](https://www.guru99.com/images/locator33.png)

**Step 4.** Change the index number to 1 so that your Target will now become document.getElementsByName("servClass")[1]. Click the Find button, and Selenium IDE should be able to highlight the "Business class" radio button, as shown below.

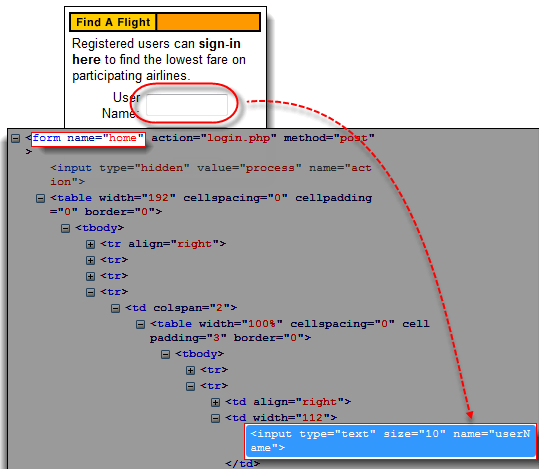
[](https://www.guru99.com/images/locator34.png)

**Locating by DOM - dom:name**

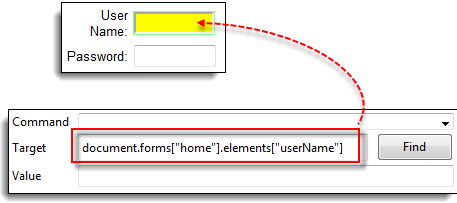
As mentioned earlier, this method will only apply if the element you are accessing is contained within a named form.

|  |  |
| --- | --- |
| **Syntax** | **Description** |
| document.forms["*name of the form*"].elements["*name of the element*"] | * name of the form = the value of the name attribute of the form tag that contains the element you want to access * name of the element = the value of the name attribute of the element you wish to access |

**Step 1.** Navigate to Mercury Tours homepage (<http://demo.guru99.com/test/newtours/>) and use Firebug to inspect the User Name text box. Notice that it is contained in a form named "home."

[](https://www.guru99.com/images/locator35.png)

**Step 2.** In Selenium IDE, type "document.forms["home"].elements["userName"]" and click the Find button. Selenium IDE must be able to access the element successfully.

[](https://www.guru99.com/images/locator36.png)

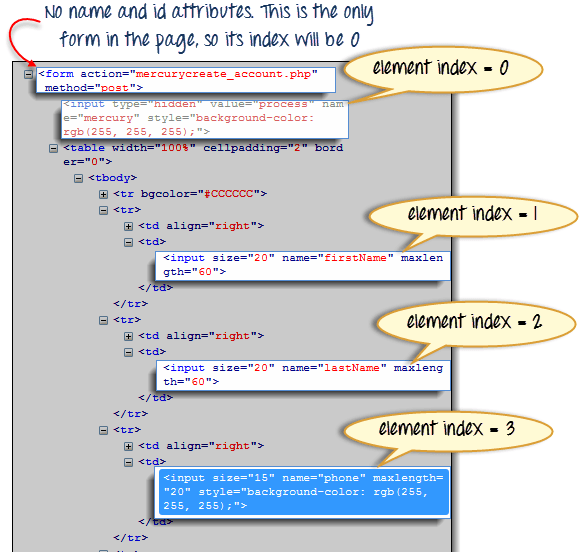
**Locating by DOM - dom:index**

This method applies even when the element is not within a named form because it uses the form's index and not its name.

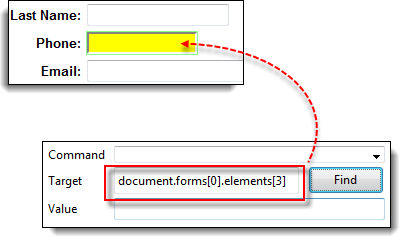
|  |  |
| --- | --- |
| **Syntax** | **Description** |
| document.forms[*index of the form*].elements[*index of the element*] | * index of the form = the index number (starting at 0) of the form with respect to the whole page * index of the element = the index number (starting at 0) of the element with respect to the whole form that contains it |

We shall access the "Phone" text box within Mercury Tours Registration page. The form in that page has no name and ID attribute, so this will make a good example.

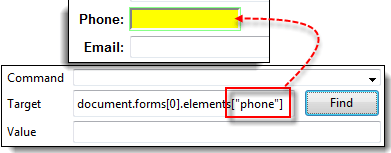
**Step 1.** Navigate to Mercury Tours Registration page and inspect the Phone text box. Notice that the form containing it has no ID and name attributes.

[](https://www.guru99.com/images/locator37.png)

**Step 2.** Enter "document.forms[0].elements[3]" in Selenium IDE's Target box and click the Find button. Selenium IDE should be able to access the Phone text box correctly.

[](https://www.guru99.com/images/locator38.png)

**Step 3.** Alternatively, you can use the element's name instead of its index and obtain the same result. Enter "document.forms[0].elements["phone"]" in Selenium IDE's Target box. The Phone text box should still become highlighted.

[](https://www.guru99.com/images/locator39.png)

**Locating by XPath**

XPath is the language used when locating XML (Extensible Markup Language) nodes. Since HTML can be thought of as an implementation of XML, we can also use [XPath](https://www.guru99.com/xpath-selenium.html) in locating HTML elements.

**Advantage:** It can access almost any element, even those without class, name, or id attributes.

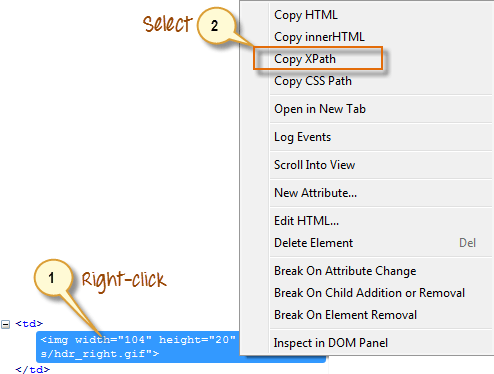
**Disadvantage:** It is the most complicated method of identifying elements because of too many different rules and considerations.

Fortunately, Firebug can automatically generate XPath locators. In the following example, we will access an image that cannot possibly be accessed through the methods we discussed earlier.

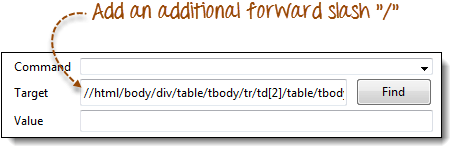
**Step 1.** Navigate to Mercury Tours Homepage and use Firebug to inspect the orange rectangle to the right of the yellow "Links" box. Refer to the image below.

[](https://www.guru99.com/images/locator40.png)

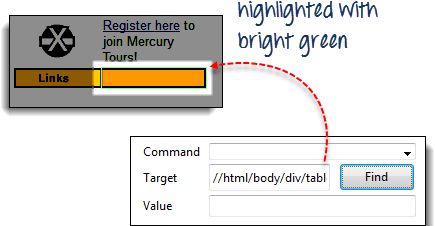
**Step 2**. Right click on the element's HTML code and then select the "Copy XPath" option.

[](https://www.guru99.com/images/locator41.png)

**Step 3.** In Selenium IDE, type one forward slash "/" in the Target box then paste the XPath that we copied in the previous step. **The entry in your Target box should now begin with two forward slashes "//".**

[](https://www.guru99.com/images/locator42.png)

**Step 4**. Click on the Find button. Selenium IDE should be able to highlight the orange box as shown below.

[](https://www.guru99.com/images/locator43.png)

**Waits, Verify Element Present/Visible in Selenium IDE**

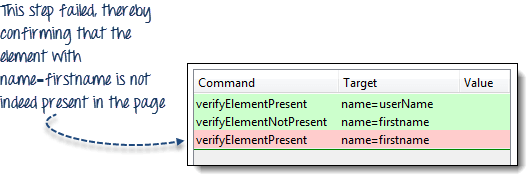
In this tutorial, we look at commands that will make your automation script more intelligent and complete.

**Verify Presence of an Element**

We can use following two commands to verify the presence of an element:

* **verifyElementPresent** - returns TRUE if the specified element was FOUND in the page; FALSE if otherwise
* **verifyElementNotPresent** - returns TRUE if the specified element was NOT FOUND anywhere in the page; FALSE if it is present.

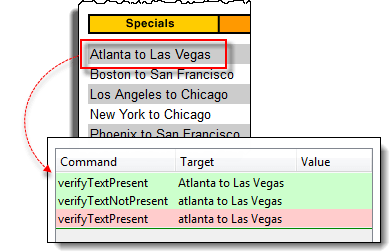
The test script below verifies that the UserName text box is present within the Mercury Tours homepage while the First Name text box is not. The First Name text box is actually an element present in the Registration page of Mercury Tours, not in the homepage. strong>Verify Presence of a Certain Text

[](https://www.guru99.com/images/image003.png)

**Verify Presence of a Certain Text**

* **verifyTextPresent** - returns TRUE if the specified text string was FOUND somewhere in the page; FALSE if otherwise
* **verifyTextNotPresent** - returns TRUE if the specified text string was NOT FOUND anywhere in the page; FALSE if it was found

Remember that these commands are case-sensitive.

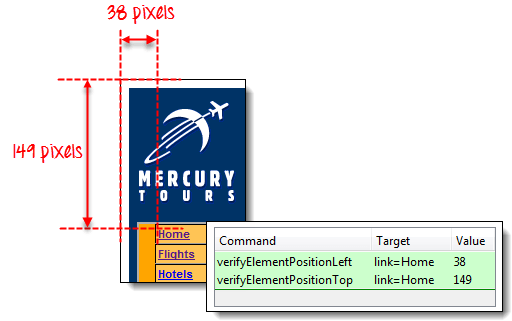
[](https://www.guru99.com/images/image004.png)

In the scenario above, "Atlanta to Las Vegas" was treated differently from "atlanta to Las Vegas" because the letter "A" of  "Atlanta" was in uppercase on the first one while lowercase on the other. When the verifyTextPresent command was used on each of them, one passed while the other failed.

**Verify Specific Position of an Element**

Selenium IDE indicates the position of an element by measuring (in pixels) how far it is from the left or top edge of the browser window.

* **verifyElementPositionLeft** - verifies if the specified number of pixels match the distance of the element from the left edge of the page. This will return FALSE if the value specified does not match the distance from the left edge.
* **verifyElementPositionTop** - verifies if the specified number of pixels match the distance of the element from the top edge of the page. This will return FALSE if the value specified does not match the distance from the top edge.

[](https://www.guru99.com/images/image005.png)

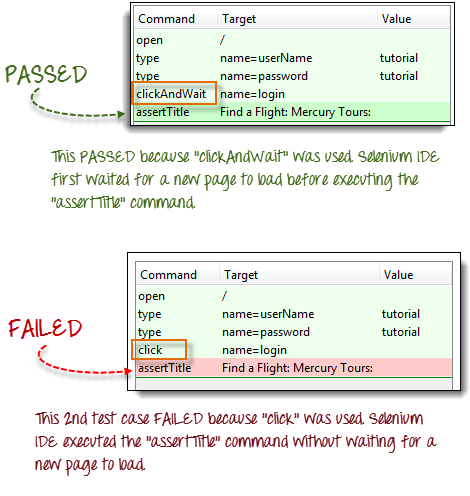
**Wait commands**

**andWait commands**

These are commands that will wait for a new page to load before moving onto the next command.

Examples are

* clickAndWait
* typeAndWait
* selectAndWait

[](https://www.guru99.com/images/image006.png)

**waitFor commands**

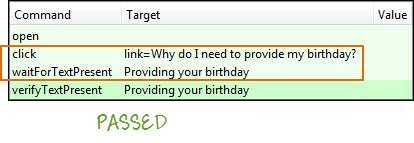
These are commands that wait for a specified condition to become true before proceeding to the next command (irrespective of loading of a new page). These commands are more appropriate to be used on AJAX-based dynamic websites that change values and elements without reloading the whole page. Examples include:

* waitForTitle
* waitForTextPresent
* waitForAlert

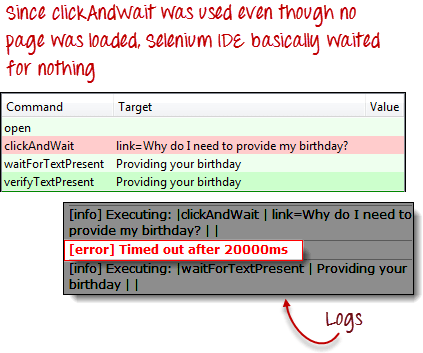
Consider the Facebook scenario below.

[](https://www.guru99.com/images/image007.png)

We can use a combination of "click" and "waitForTextPresent" to verify the presence of the text "Providing your birthday."

[](https://www.guru99.com/images/image008.png)

We cannot use clickAndWait because no page was loaded upon clicking on the "Why do I need to provide my birthday?" link. If we do, the test will fail

[](https://www.guru99.com/images/image009.png)