**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 CSS Locator in Selenium IDE

**What You Will Learn:**

There are commands that do not need a locator (such as the “open” command). However, most of them do need element Locators in Selenium webdriver.

**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 element locators in Selenium webdriver, based on your application support.

**Locating by ID**

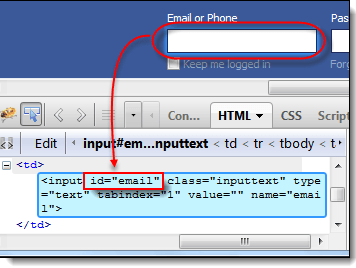
This is the most common way of locating elements since ID’s are supposed to be unique for each element.

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

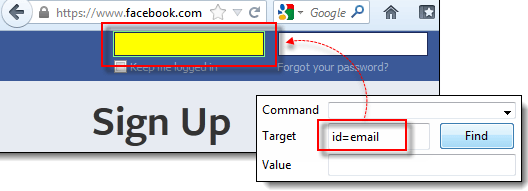
Skip Ad

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.”



**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.



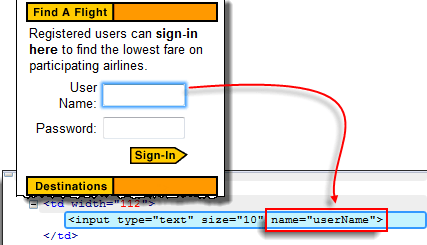
**Locating by Name**

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*

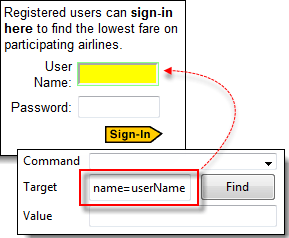
In the following demonstration, we will now use Mercury Tours because all significant elements have names.

**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.



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.



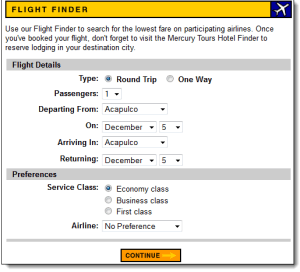
**Locating by Name using Filters**

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

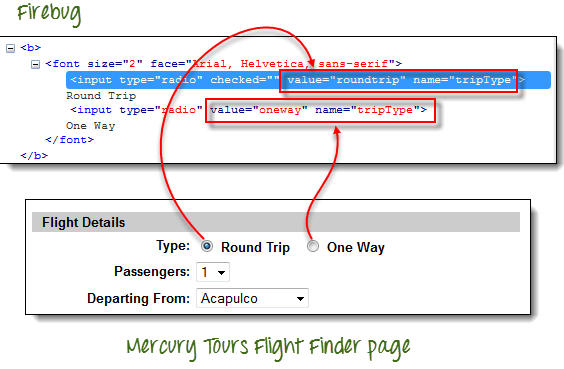
**Target Format**: name=*name\_of\_the\_element* *filter*=*value\_of\_filter*

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.

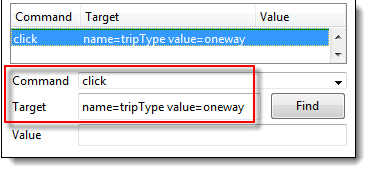


**Step 2.** Using Firebug, notice that the Round Trip and One Way radio buttons have the same name “tripType.” However, they have different VALUE attributes so we can use each of them as our filter.

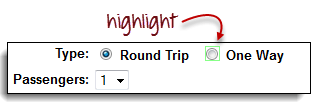


**Step 3.**

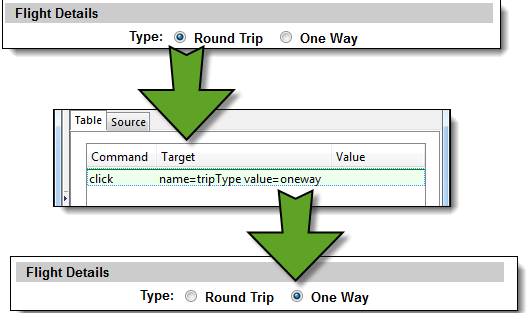
* We are going to access the One Way radio button first. Click the first line on the Editor.
* In the Command box of Selenium IDE, enter the command “click”.
* In the Target box, enter “name=tripType value=oneway”.  The “value=oneway” portion is our filter.



**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.



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



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 CSS locator in Selenium 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*

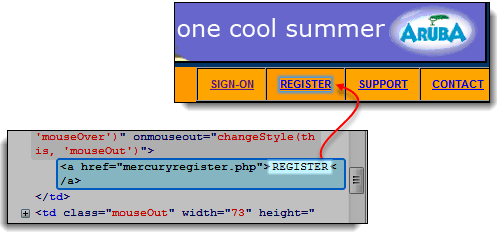
In this example, we shall access the “REGISTER” link found on the Mercury Tours homepage.

**Step 1.**

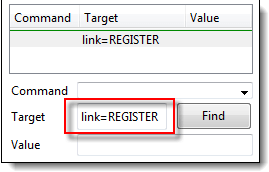
* First, make sure that you are logged off from Mercury Tours.
* Go to Mercury Tours homepage.

**Step 2**.

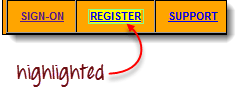
* Using Firebug, inspect the “REGISTER” link. The link text is found between and tags.
* In this case, our link text is “REGISTER”. Copy the link text.



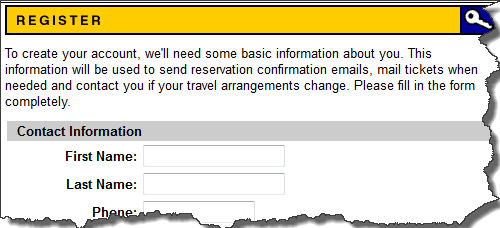
**Step 3**. Copy the link text in Firebug and paste it onto Selenium IDE’s Target box. Prefix it with “link=”.



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



**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.



**Locating by CSS Selector**

CSS Selectors in Selenium are string patterns used to identify an element based on a combination of HTML tag, id, class, and attributes. Locating by CSS Selectors in Selenium 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 in Selenium 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**

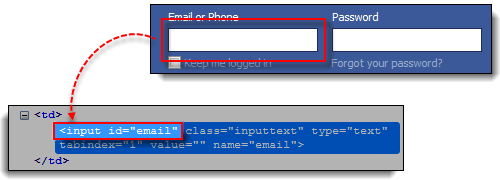
Again, we will use Facebook’s Email text box in this example. As you can remember, it has an ID of “email,” and we have already accessed it in the “Locating by ID” section. This time, we will use a Selenium CSS Selector with ID in accessing that very same element.

|  |  |
| --- | --- |
| **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 Selenium CSS Selector with ID * id = the ID of the element being accessed |

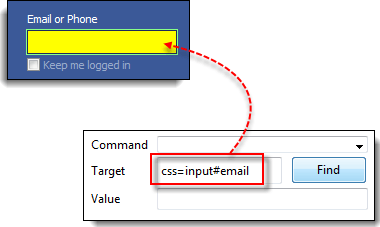
**Keep in mind that the ID is always preceded by a hash sign (#).**

**Step 1.** Navigate to [www.facebook.com](https://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”.



**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.

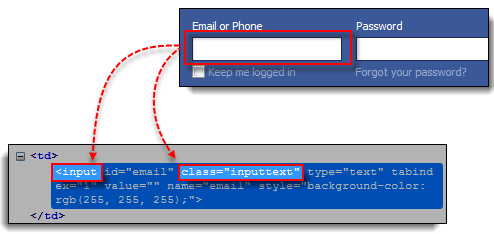


**Locating by CSS Selector – Tag and Class**

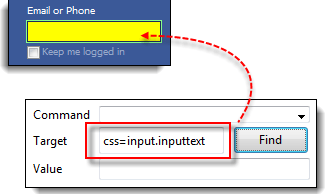
Locating by CSS Selector in Selenium using an HTML tag and a class name is similar to using a tag and ID, but in this case, a dot (.) is used instead of a hash sign.

|  |  |
| --- | --- |
| **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 |

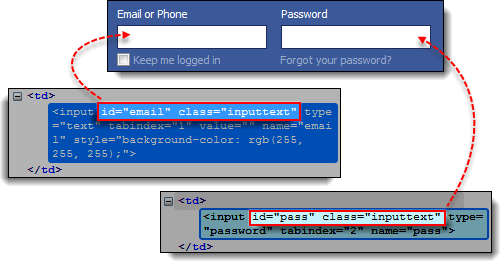
**Step 1.** Go to the demo page http://demo.guru99.com/test/facebook.html and use Firebug to inspect the “Email or Phone” text box. Notice that its HTML tag is “input” and its class is “inputtext.”



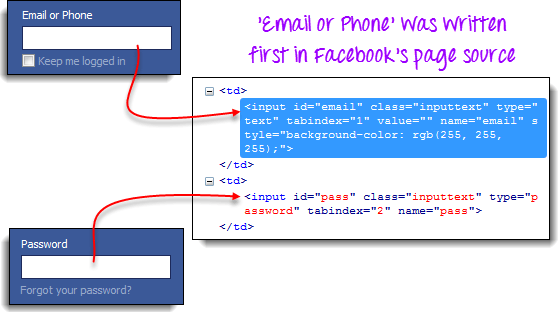
**Step 2.** In Selenium IDE, enter “css=input.inputtext” in the Target box and click Find. Selenium IDE should be able to recognize the Email or Phone text box.



**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.



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.

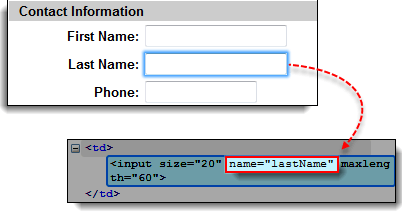


**Locating by CSS Selector – Tag and Attribute**

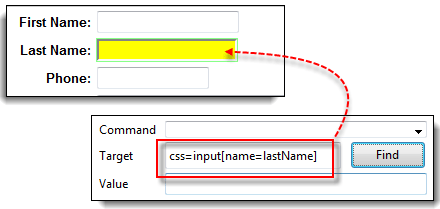
This strategy uses the HTML tag and a specific attribute of the element to be accessed.

|  |  |
| --- | --- |
| **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”).



**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.

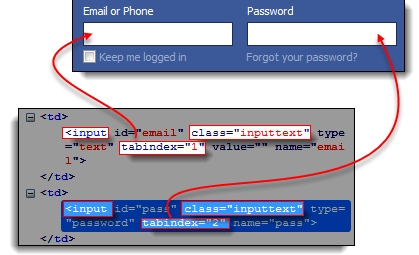


**When multiple elements have the same HTML tag and attribute, only the first one will be recognized**. This behavior is similar to locating elements using CSS selectors with the same tag and class.

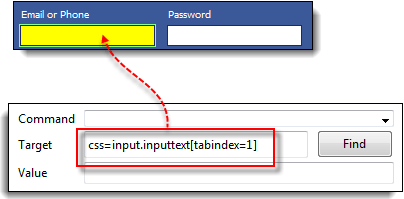
**Locating by CSS Selector – tag, class, and attribute**

|  |  |
| --- | --- |
| **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. |

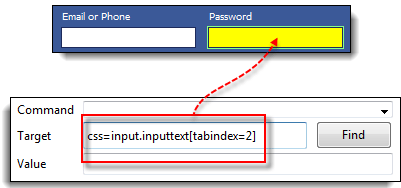
**Step 1.** Go to the demo page http://demo.guru99.com/test/facebook.html and use Firebug to inspect the ‘Email or Phone’ and ‘Password’ input boxes. Take note of their HTML tag, class, and attributes. For this example, we will select their ‘tabindex’ attributes.



**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.



**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.

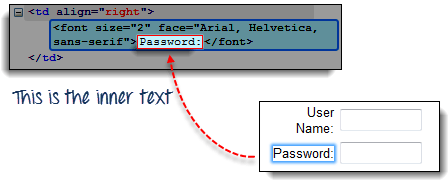


**Locating by CSS Selector – inner text**

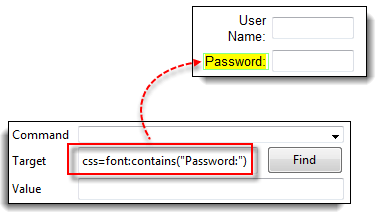
As you may have noticed, HTML labels are seldom given id, name, or class attributes. So, how do we access them? The answer is through the use of their inner texts. **Inner texts are the actual string patterns that the HTML label shows on the page.**

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

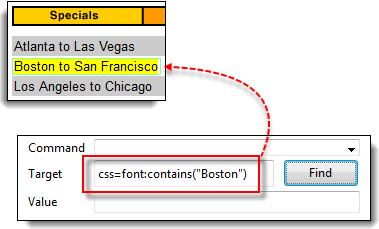
**Step 1.** Navigate to Mercury Tours’ homepage http://demo.guru99.com/test/newtours/ and use Firebug to investigate the “Password” label. Take note of its HTML tag (which is “font” in this case) and notice that it has no class, id, or name attributes.



**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.



**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.



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

The [Document Object Model (DOM)](https://www.guru99.com/understanding-dom-fool-guide.html), 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 in Selenium anyway.

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

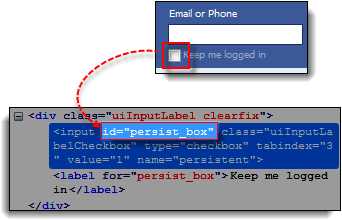
* getElementById
* getElementsByName
* dom:name (applies only to elements within a named form)
* dom:index

**Locating by DOM – getElementById**

Let us focus on the first method – using the getElementById method of DOM in Selenium. 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.



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.



**Locating by DOM – getElementsByName**

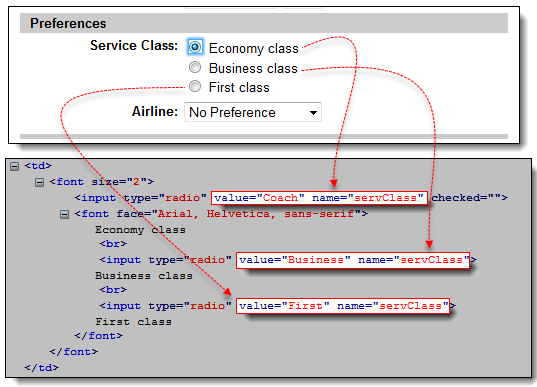
The getElementById method can access only one element at a time, and that is the element with the ID that you specified. The getElementsByName method is different. It collects an array of elements that have the name that you specified. You access the individual elements using an index which starts at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

|  |  |
| --- | --- |
| How to use Locators in Selenium IDE | **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 | **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. |

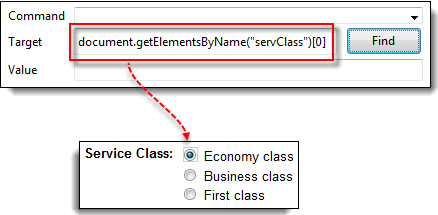
|  |  |
| --- | --- |
| **Syntax** | **Description** |
| document.getElementsByName(“*name*“)[*index*] | * name = name of the element as defined by its ‘name’ attribute * index = an integer that indicates which element within getElementsByName’s array will be used. |

**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.

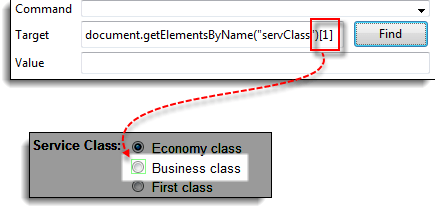
**Step 2.** Using Firebug, inspect the three radio buttons at the bottom portion of the page (Economy class, Business class, and First class radio buttons). Notice that they all have the same name which is “servClass”.



**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.



**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.

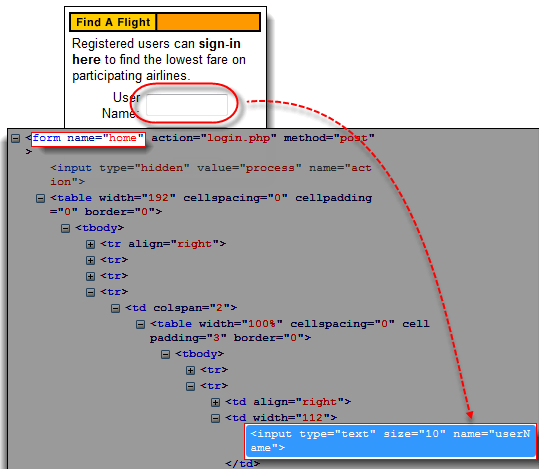


**Locating by DOM – dom:name**

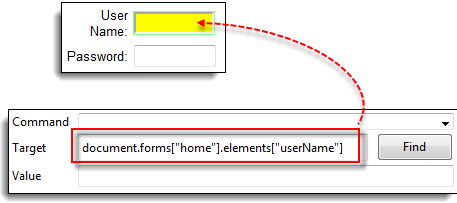
As mentioned earlier, this method will only apply if the element you are accessing is contained within a named form.

|  |  |
| --- | --- |
| **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.”



**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.



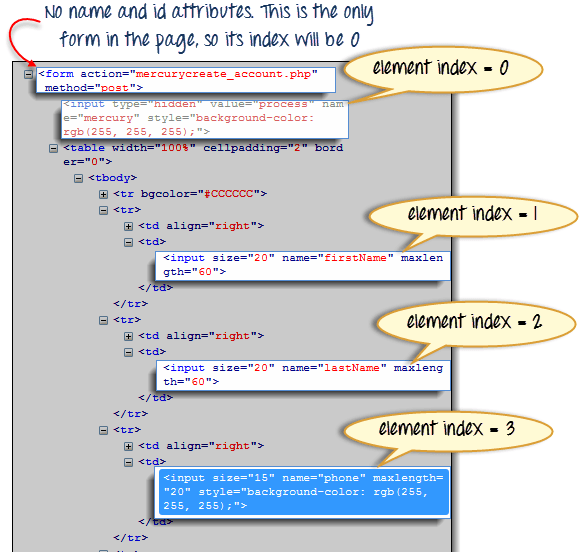
**Locating by DOM – dom:index**

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

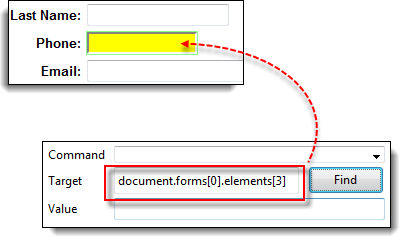
|  |  |
| --- | --- |
| **Syntax** | **Description** |
| document.forms[*index of the form*].elements[*index of the element*] | * index of the form = the index number (starting at 0) of the form with respect to the whole page * index of the element = the index number (starting at 0) of the element with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

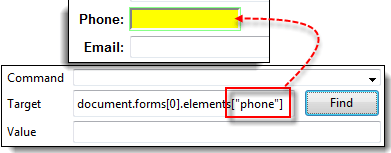
**Step 1.** Navigate to Mercury Tours Registration page and inspect the Phone text box. Notice that the form containing it has no ID and name attributes.



**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.



**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.



**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.

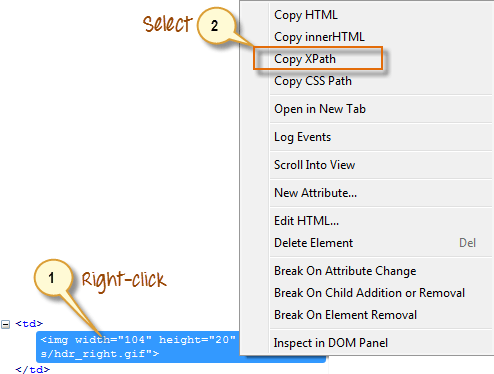
**Disadvantage:** It is the most complicated method of identifying elements because of too many different rules and considerations.

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

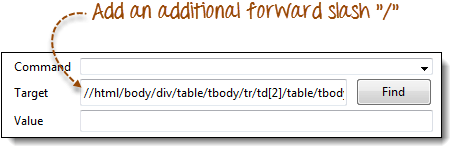
**Step 1.** Navigate to Mercury Tours Homepage and use Firebug to inspect the orange rectangle to the right of the yellow “Links” box. Refer to the image below.



**Step 2**. Right click on the element’s HTML code and then select the “Copy XPath” option.



**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 “//”.**



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

