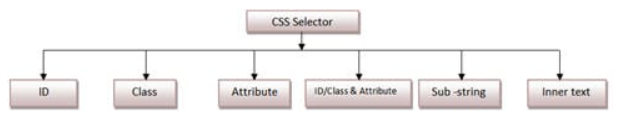
Six types of locators for identifying web elements on a web page

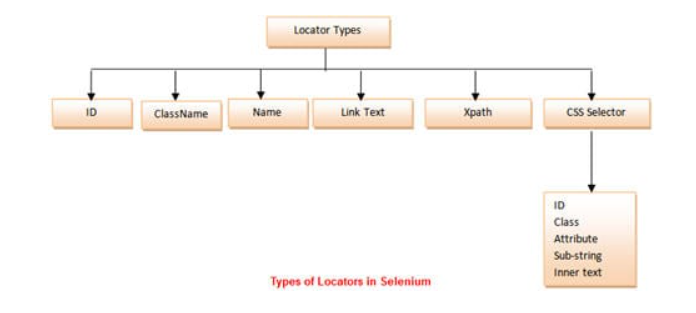
* ID
* ClassName
* Name
* Link Text
* XPath
* css selector

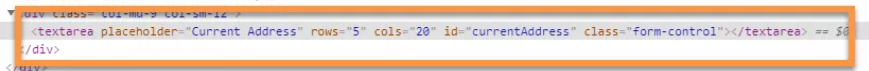
CSS Selector is the combination of an element selector and a selector value which identifies the web element within a web page. The composite of an element selector and selector value is known as **Selector Pattern**.

Selector Pattern is constructed using **HTML tags, attributes and their values**. The central theme behind the procedure to create CSS Selector and Xpath are very much similar underlying the only difference in their construction protocol.

Like Xpath, **CSS selector** can also locate web elements having no ID, class or Name.







***How to Combine the ID and other Attributes of the web element to create a CSS Selector?***

In the given element, the *HTML* structure contains a *textarea tag, id, and placeholder attribute*. We will use these together to create a *CSS Selector* statement that can easily recognize that element. So, the *CSS*

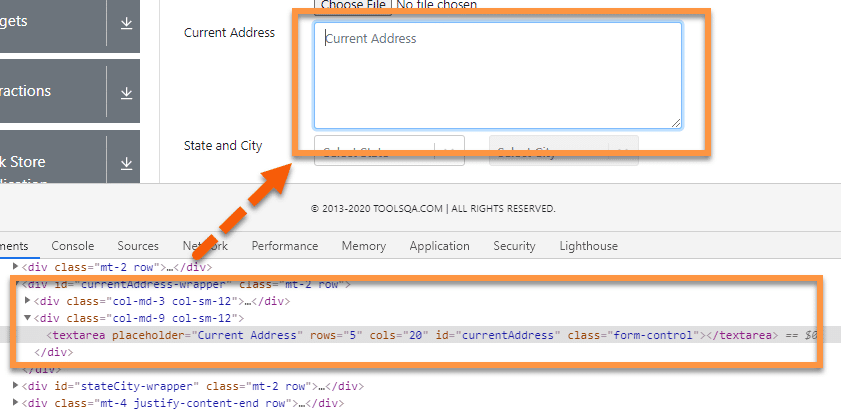
**“**textarea**#currentAddress”**  
**“textarea.form-control”**

**“**textarea**[placeholder='Current Address']”**

#### How to locate a web element when one element is the **direct parent/child** of another element?

CSS Selectors allow you to select an element by using the locator of the parent element and then moving to the child element. The CSS Selector for locating the child element can be syntactically represented as follows:

|  |
| --- |
| **Parent\_locator > child\_locator** |



In the above example, we have a “**textarea**” HTML tag enclosed in the bracket, which is the child tag of “***div.”*** Assume a scenario where we are not able to identify the “**textarea**” by using its attributes, but we can identify its parent HTML tag, then we can use it to access the child tag. Let’s create the CSS selector for locating the ***textarea*** element:

|  |
| --- |
| **div>textarea[placeholder='Current Address']** |

Here we have first used the ***locator for a parent***then “**>**” followed by ***the child locator***. Similarly, this can be extended to the sub child also by adding another “**>**” followed by another locator.

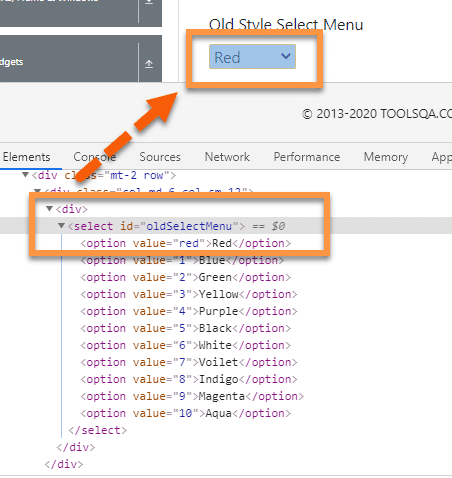
#### How to locate a web element when the element exists in the hierarchy?

Similar to the child and sub-child, we can also use a CSS Selector to select the ***nth-child of an HTML tag***. It is quite useful in recognizing list elements or in scenarios where a parent has multiple child elements with non-consistent attributes.

The syntax for locating the nth-child will be:

**Parent CSS locator >** **Child HTML tag** **: nth-of-type**(index)

Selecting nth-child using CSS Selector, for this, we will be using the following site link: [***https://www.demoqa.com/select-menu***](https://www.demoqa.com/select-menu)***.***



Let’s take the above example; we will try to find the ***CSS Selector***for the child element of “**ul**” HTML tag, i.e.    “**li”** Say, we want to find the 2nd child element of the “**ul”** then the CSS Selector expression for the same will be:

|  |  |
| --- | --- |
|  | **Parent CSS locator >** **Child HTML tag** **: nth-of-type**(index)  **select#oldSelectMenu>option:nth-of-type**(2) |

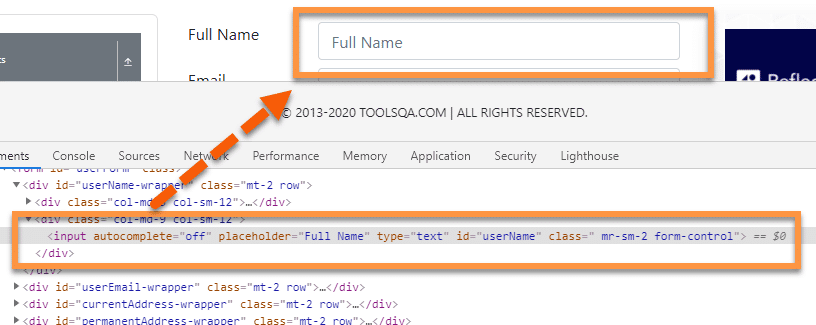
Here we started with the parent **CSS Selector** tag, followed by “**>”** which is followed by the ***HTML***  tag of the child. The child ***HTML tag is then followed by id symbol –***“***:*** ”, which is followed by “***nth-of-type(index)”*** where bracket accepts the index of the required element.

So this way, we can locate any of the HTML elements in the hierarchy.

### *****How to locate a web element using text strings?*****

Similar to XPath, ***CSS Selector*** also allows users to locate elements by using partial strings. It uses different symbols to represent the start, end, and the contents inside a text. Let’s have a look at some examples to understand more about ***CSS Sub-Strings*** in detail.

For example, we will use the following element, i.e., “**Full Name,**” as marked in the below image for locating the text box on the page.



#### How to locate a web element using the starting text?

We can locate an element by using the starting text of the element. It is quite useful if you know the starting text of the element attribute. We can use the starting character sequence of the attribute value to locate the element using CSS Selectors.

The Symbol for representing the *starting text* of a string is: **‘^’**

Using this symbol in the *CSS Selector*, the expression for locating the web element will be

|  |  |
| --- | --- |
|  | input[**id^=**'userN'] |

Here, we have used the ***id*** attribute. The value of the *id* attribute in the *HTML* is “***userName.”*** In this expression, we have used the ***first five characters*** of the expression. We can use any number of characters from the beginning.

***How to locate a web element using the* Ending text?**

Similar to the starting text, we can also use ***ending text***to recognize the element. A sequence of the ending character of the attribute value can locate any web element.

The Symbol for representing the *ending text* of a string is: ‘$’

Using this symbol in the *CSS Selector*, the expression for locating the web element will be:

|  |  |
| --- | --- |
|  | input[**id$=**'ame'] |

Here, again we have used the ***id*** attribute whose value is “***username***”. Here we have used the *last three characters* of the attribute value. The attribute and value can change as per the scenario.

***How to locate a web element using the* contains text?**

Other than *starting and ending*, the *CSS Selector* in Selenium is also available **with contains text().** It can locate the element by using *any sequential characters* from the attribute value.

The Symbol for representing the *contains the text*: **‘\*’**

Using the same symbol in *CSS Selector*, the expression for the above elements will be:

|  |  |
| --- | --- |
|  | input[**id\*=**'erNa'] |

Here, we have used the ***middle characters***of the *id* attribute value, i.e., “***username***” from the *HTML* to locate the element.

**Xpath selector**

XPath, also known as XML Path Language, is a language for selecting elements from an XML document. As HTML and XML follow the same document structure, XPath can also be used to select elements from a web page.

**Absolute path selector**.

"/html/body/div[1]/div[1]/div[2]/div[1]/div/div/h3[2]"

**Relative XPath selector**

As you can see, the XPath selector **//div/p** selects the **p** tag inside the **div** tag. This is an example of a relative XPath selector.

**Relative XPath selector starts with //**. Then you specify the structure of the element you want to select. In this case, **div/p**.

So, **//div/p** means select the **p**element inside a **div** element, does not matter what comes before it.

<!DOCTYPE html>  
<[html](http://december.com/html/4/element/html.html) lang="en">  
<[head](http://december.com/html/4/element/head.html)>  
    <[meta](http://december.com/html/4/element/meta.html) charset="UTF-8">  
    <[meta](http://december.com/html/4/element/meta.html) name="viewport" content="width=device-width, initial-scale=1.0">  
    <[title](http://december.com/html/4/element/title.html)>Basic HTML Document</[title](http://december.com/html/4/element/title.html)>  
</[head](http://december.com/html/4/element/head.html)>  
<[body](http://december.com/html/4/element/body.html)>  
    <[h1](http://december.com/html/4/element/h1.html)>Hello World</[h1](http://december.com/html/4/element/h1.html)>  
  
    <[div](http://december.com/html/4/element/div.html)>  
        <[p](http://december.com/html/4/element/p.html)>this is message</[p](http://december.com/html/4/element/p.html)>  
    </[div](http://december.com/html/4/element/div.html)>  
  
    <[div](http://december.com/html/4/element/div.html)>  
        <[span](http://december.com/html/4/element/span.html)>hello world</[span](http://december.com/html/4/element/span.html)>  
    </[div](http://december.com/html/4/element/div.html)>  
</[body](http://december.com/html/4/element/body.html)>  
</[html](http://december.com/html/4/element/html.html)>

As you can see, the XPath selector **//div/p** selects the **p** tag inside the **div** tag. This is an example of a relative XPath selector.

Relative XPath selector starts with **//**. Then you specify the structure of the element you want to select. In this case, **div/p**.

So, **//div/p** means select the **p**element inside a **div** element, does not matter what comes before it.

You can also select elements by different attributes like **id**, **class**, **type,** etc. **using XPath selector**. Let’s see how to do that.

<!DOCTYPE html>  
<[html](http://december.com/html/4/element/html.html) lang="en">  
<[head](http://december.com/html/4/element/head.html)>  
    <[meta](http://december.com/html/4/element/meta.html) charset="UTF-8">  
    <[meta](http://december.com/html/4/element/meta.html) name="viewport" content="width=device-width, initial-scale=1.0">  
    <[title](http://december.com/html/4/element/title.html)>Basic HTML Document</[title](http://december.com/html/4/element/title.html)>  
</[head](http://december.com/html/4/element/head.html)>  
<[body](http://december.com/html/4/element/body.html)>  
    <[h1](http://december.com/html/4/element/h1.html)>Hello World</[h1](http://december.com/html/4/element/h1.html)>  
    <[**div**](http://december.com/html/4/element/div.html)**class="container1"**>  
        <[p](http://december.com/html/4/element/p.html)>this is message</[p](http://december.com/html/4/element/p.html)>  
        <[span](http://december.com/html/4/element/span.html)>this is another message</[span](http://december.com/html/4/element/span.html)>  
    </[div](http://december.com/html/4/element/div.html)>  
    <[**div**](http://december.com/html/4/element/div.html)**class="container1"**>  
        <[h2](http://december.com/html/4/element/h2.html)>heading 2</[h2](http://december.com/html/4/element/h2.html)>  
        <[p](http://december.com/html/4/element/p.html)>Lorem ipsum dolor sit amet consectetur, adipisicing elit. Quibusdam  
eligendi doloribus sapiente, molestias quos quae non nam incidunt quis delectus  
facilis magni officiis alias neque atque fuga? Unde, aut natus?</[p](http://december.com/html/4/element/p.html)>  
    </[div](http://december.com/html/4/element/div.html)>  
    <footer>  
        **<**[**span**](http://december.com/html/4/element/span.html)**id="footer-msg">this is a footer</**[**span**](http://december.com/html/4/element/span.html)>  
    </footer>  
</[body](http://december.com/html/4/element/body.html)>  
</[html](http://december.com/html/4/element/html.html)>

Let’s say you want to select all the **div** elements which have the **class** name **container1**. To do that, you can use the XPath selector **//div[@class=’container1′]**

As you can see, I have 2 elements which match the XPath selector **//div[@class=’container1′]**

To select the first **div** element with the **class** name **container1**, add **[1]** at the end of the XPath select, as shown in the screenshot below.

The same way, you can select the second **div** element with the **class** name **container1** using the XPath selector **//div[@class=’container1′][2]**

You can select elements by **id** as well.

For example, to select the element which has the **id** of **footer-msg**, you can use the XPath selector **//\*[@id=’footer-msg’]**

Here, the **\*** before **[@id=’footer-msg’]**is used to **select any element regardless of their tag.**

Relative xpath – This path begins from any part of the DOM html. It is represented by double slash // and helps to identify elements from any part of the web page and the xpath expression is not a lengthy one.

It is always better to use relative xpath since we need not start from the root to locate our element. So in case any attributes or their values are getting modified, the probability of getting affected to an incorrect xpath is less.

**Syntax** −

driver. find\_element\_by\_xpath("//input[@class='gsc-input']")

Absolute xpath – This path starts from the root html to the desired element. It is represented by single slash /. It is lengthier compared to relative xpath and if any of the attributes or their values starting from the root element get changed, then we end up not being able to uniquely identify our desired object on the page.

**Syntax** −

driver. find\_element\_by\_xpath("/html/body/div/input")

**Xpath can perform bidirectional flow which means the traversal can be both ways from parent to child and vice- versa**.

## Parent to child

**Syntax** −

driver.find\_element\_by\_xpath("**//table/tbody/tr[2]/td[2]")**

## Child to parent

**Syntax** −

driver. find\_element\_by\_xpath("**//input[@id='job']/parent::div**")

In terms of speed, xpath is slower compared to css.

**There are some differences between xpath and css listed below** −

* **Xpath allows bidirectional flow** which means the traversal can be both ways from parent to child and child to parent as well. Css allows only one directional flow which means the traversal is from parent to child only.
* **Xpath is slower** in terms of performance and speed. Css has better performance and speed than xpath.
* **Xpath** allows identification with **the help of visible text** appearing on screen with the help of **text() function**. Css does not have this feature.
* **Customized css** can be created directly with the help of attributes id and class. For id, the css expression is represented by **#** followed by the id [ #<<id expression>>. For class, the css expression is represented by **.** followed by the class [.<<class expression>>]. Xpath does not have any feature like this.
* Xpath expression is represented by [**//**tagname[**@**attribute = 'value']. The css expression is repression is represented by [tagname[attribute = 'value'].
* **There are two types of xpath – absolute and relative. But css has no such types**.

## **Node Order**

* Every node in our DOM is numbered. If we have a list ul which contains five li items, we can access each of those items specifically by indexing from 1 to 5:
* <ul>  
   <li>London</li>  
   <li>Miami</li>  
   <li>New Dehli</li>  
  </li>
* If we query //ul/li[1] we will return <li>London</li> — note that values are **not**zero-indexed. XPath indexing begins at 1.

//\*[[@id](http://twitter.com/id)="\_obv.shell.\_surface\_1600536527994"]/**div/div[1]/div[2] /div[1]/div[1]/a/span[1]**

/html/body/div[1]/div[2]/**div/div[1]/div[2]/div[1]/div[1]/a/span[1]**

WHY XPATH’S

While trying to write your Selenium scripts, you must have noticed that you need to identify the elements. Selenium gives you the flexibility to identify the elements via various mechanism like ID, Name, Tagname, CSS Selector etc. But in most of the scenarios and in most of the webpages, not every element will be having a unique name and id. **Though you can use CSS Selector to select that element, but xpath is a much easier (but complicated) way to identify element**.

**Absolute XPath**

An Absolute Xpath is the one that starts from the root node. A good way to recognize this kind of element is that it starts with a forward slash ( / ). Absolute xpath is very fast, since it is a step by step, thorough path provided. The downside, however, is that any single change in the document tree structure would make the path un-usable, after the part where the new change has been made.

Example of an absolute xpath can be

/html/head/body/table/tbody/tr/th

Now suppose, if I were to add any new node in between , suppose a div tag somewhere in between

/html/head/body/table/div/tbody/tr/th

Now, the document structure has changed. So if you are using the xpath up to the <th> tag, then it would result in an error and you have to change the xpath again, to use the new structure.

**Relative Xpath**

A relative xpath doens’t starts from the root node. It starts from the node that you want it to start with. An example of a relative xpath would generally start with a double slash ( // ).

An example would be

//table/tbody/tr/th

Relative xpaths are relatively slower, as we give it a partial path to select the elements from, not the full path, as we do in Absolute xpath.

**While working with relative xpaths, you will many times see xpaths using different attributes to identify an element**.

For example, you can use any attribute to find an xpath using the form

**//tagname[@attribute-name='value1']**

If you want to use multiple attributes to form an xpath, you need to use

//tagname**[@attribute1='value'][@attribute2='value']**

We can also use the contains( ) method, to form an xpath, in case, actual matching or an attribute cannot be done, or if you have multiple childs of a node.

//tagname[**contains(@attribute,'value1')**]

We can also use the starts-with( ) method to form an xpath, like this

//tagname[**starts-with(@attribute-name,'value1')**]

You can use the following node method to select the **very next element of a node** like

xpath/**following::**xpath-of-the-element

e.g –  //input[@id='']**/following::input[1]**

To select an element just above a node, use the preceding method

xpath/**preceding**::xpath-of-the-element

e.g –  //input[@id='']/ **preceding::input[1]**

Alternatively, we can copy the full XPath, which provides the full path from the root to our element. For our span element this looks like:

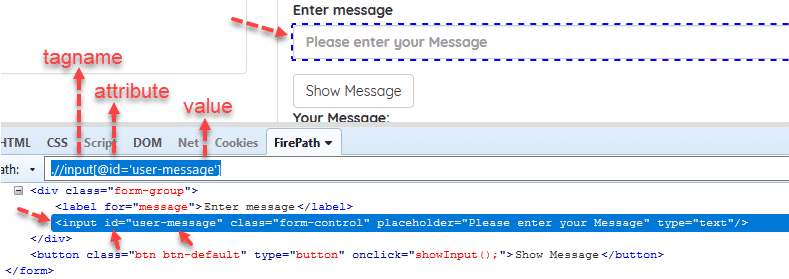
/html/body/div[1]/div[2]/**div/div[1]/div[2]/div[1]/div[1]/a/span[1]**

## **XPath Selenium Selectors**

We can find the location of any element on a web page using XML path expressions. The basic syntax for XPath is shown below:

**Syntax = //tagname[@attribute=’Value‘]**

**Example = //input[@id=’user-message‘]**

[](https://224926-685269-raikfcquaxqncofqfm.stackpathdns.com/wp-content/uploads/2017/09/xpath-selenium-tag-value-attribute.png)

You can also use **class**, **name**, **link text**, and the other attributes to locate an element with XPath as shown above.

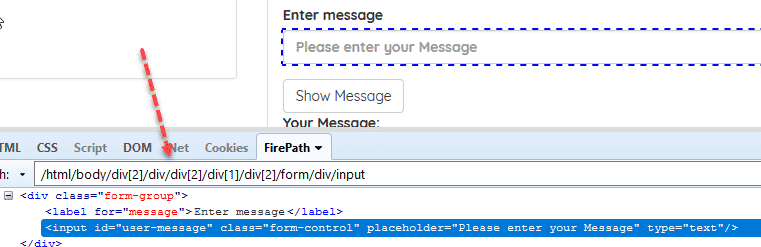
## **Absolute and Relative XPath**

Generally, in some test automation engineer interviews, I asked the difference between absolute and relative XPath. Actually, it is the answer is very easy.

### ****Absolute XPath****

* It is a direct way to locate an element.
* It is very brittle.
* Starts with single slash “/” that means starting to search from the root.

**Example:**/html/body/div[2]/div/div[2]/div[1]/div[2]/form/div/input

[](https://224926-685269-raikfcquaxqncofqfm.stackpathdns.com/wp-content/uploads/2017/09/xpath-in-selneium-absolute-xpath.png)

### ****Relative XPath****

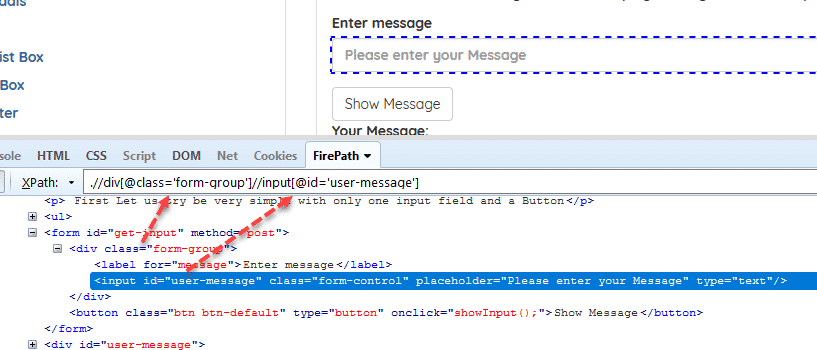
* Starts from the middle of the HTML DOM.
* Starts with a double slash “//” that means it can start to search anywhere in the DOM structure.
* Shorter than Absolute XPath.
* Less fragile.

**Example:**//div[@class=’form-group’]**//**input[@id=’user-message’]

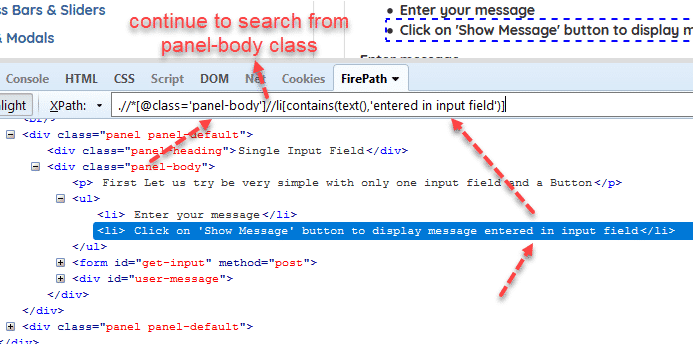
### ****Relative XPath****

* Starts from the middle of the HTML DOM.
* Starts with a double slash “//” that means it can start to search anywhere in the DOM structure.
* Shorter than Absolute XPath.
* Less fragile.

**Example:**//div[@class=’form-group’]//input[@id=’user-message’]

[](https://224926-685269-raikfcquaxqncofqfm.stackpathdns.com/wp-content/uploads/2017/09/relative-xpath.png)

**Example:**//\*[@class=’panel-body’]//li[contains(text(),’entered in input field’)]

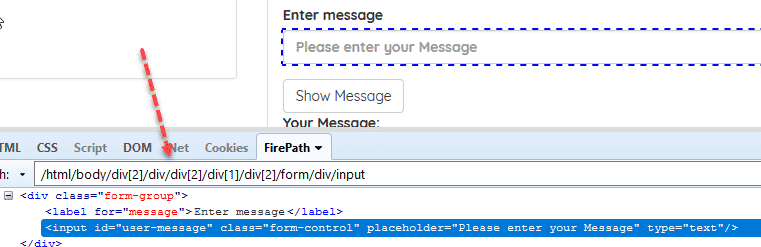
[](https://224926-685269-raikfcquaxqncofqfm.stackpathdns.com/wp-content/uploads/2017/09/relative-xpath-2.png)

## **Writing Smart XPaths for Complex and Dynamic Elements**

### ****Tag – Attribute – Value Trio****

**Syntax: //tag[@attribute=’value‘]**

**Example:**//input[@id, ‘user-message’]

[](https://224926-685269-raikfcquaxqncofqfm.stackpathdns.com/wp-content/uploads/2017/09/xpath-in-selneium-absolute-xpath.png)

**Examples:**

Java

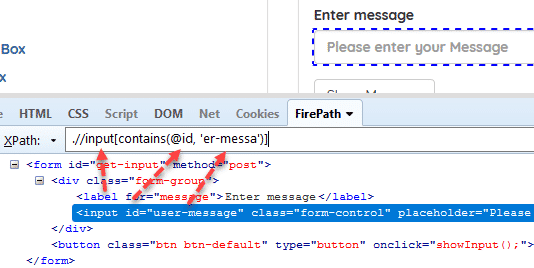
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | //input[@type='send text']    //label[@id='clkBtn']    //input[@value='SEND']    //\*[@class='swtestacademy']  --> "\*" means, search "swtestacademy" class for all tags.    //a[@href='http://www.swtestacademy.com/']    //img[@src='cdn.medianova.com/images/img\_59c4334feaa6d.png'] |

### ****Contains****

It is very handy XPath Seleniumlocator and sometimes it saves the life of a test automation engineer. When an attribute of an element is dynamic, then you can use contains() for the constant part of the web element but also you can use contains() in any condition when you need.

**Syntax: //tag[contains(@attribute, ‘value‘)]**

**Example:**//input[contains(@id, ‘er-messa’)]

[](https://224926-685269-raikfcquaxqncofqfm.stackpathdns.com/wp-content/uploads/2017/09/xpath-contains.png)

**Examples:**

Java

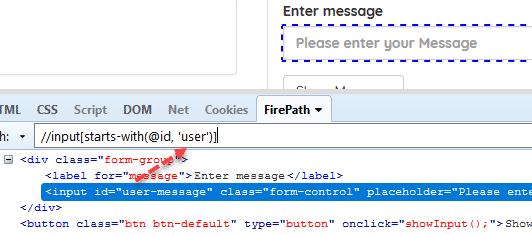
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | //\*[contains(@name,'btnClk')]  --> It searches "btnClk" for all name attributes in the DOM.    //\*[contains(text(),'here')]  --> It searches the text "here" in the DOM.    //\*[contains(@href,'swtestacademy.com')]  --> It searches "swtestacademy.com" link in the DOM. |

### ****Starts-with****

This method checks the starting text of an attribute. It is very handy to use when the attribute value changes dynamically but also you can use this method for non-changing attribute values.

**Syntax: //tag[starts-with(@attribute, ‘value‘)]**

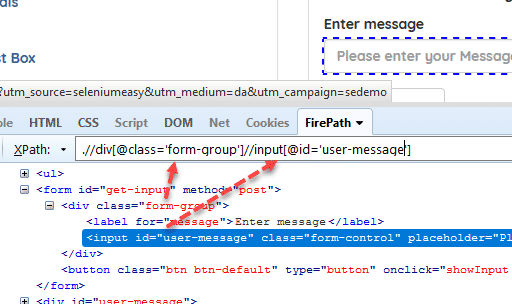
**Example:**//input[starts-with(@id, ‘user’)]

[](https://224926-685269-raikfcquaxqncofqfm.stackpathdns.com/wp-content/uploads/2017/09/xpath-starts-with.png)

### ****Chained Declarations****

We can chain multiple relative XPath declarations with **“//” double slash** to find an element location as shown below.

**Example:**//div[@class=’form-group’]//input[@id=’user-message’]

[](https://224926-685269-raikfcquaxqncofqfm.stackpathdns.com/wp-content/uploads/2017/09/chained-xpath.png)

### ****Operator “or”****

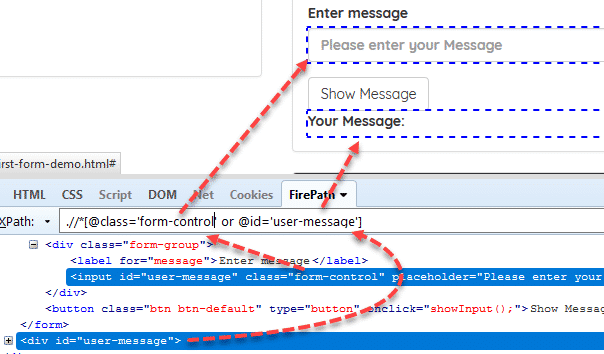
In this method, we use two interrogation conditions such as A and B and return a result-set as shown below:

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **Result** |
| False | False | No Element |
| True | False | Returns A |
| False | True | Returns B |
| True | True | Returns Both |

**“or” is case-sensitive**, you should not use capital “OR”.

**Syntax: //tag[XPath Statement-1 or XPath Statement-2]**

**Example:**//\*[@id=’user-message’ or @class=’form-control’]

[](https://224926-685269-raikfcquaxqncofqfm.stackpathdns.com/wp-content/uploads/2017/09/xpath-or-statement.png)

### ****Operator “and”****

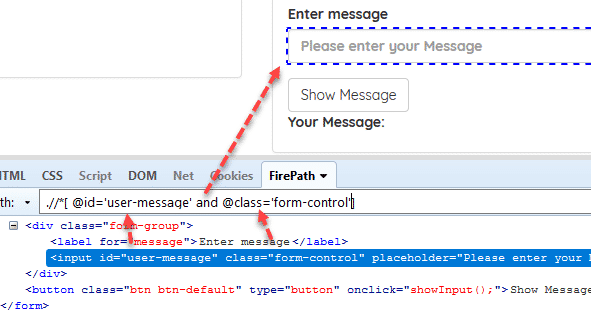
In this method, we use two interrogation conditions such as A and B and return a result-set as shown below:

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **Result** |
| False | False | No Element |
| True | False | No Element |
| False | True | No Element |
| True | True | Returns Both |

**“and” is case-sensitive**, you should not use capital “AND”.

**Syntax: //tag[XPath Statement-1 and XPath Statement-2]**

**Example:**//\*[@id=’user-message’ and @class=’form-control’]

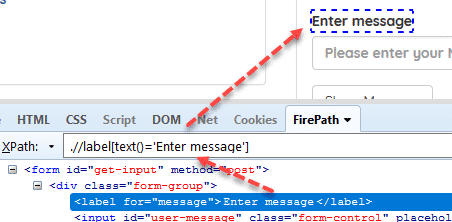
[](https://224926-685269-raikfcquaxqncofqfm.stackpathdns.com/wp-content/uploads/2017/09/xpath-and-statement.png)

### ****Text****

We can find an element with its exact text.

**Syntax: //tag[text()=’text value‘]**

**Example:**.//label[text()=’Enter message’]

[](https://224926-685269-raikfcquaxqncofqfm.stackpathdns.com/wp-content/uploads/2017/09/xpath-text.png)

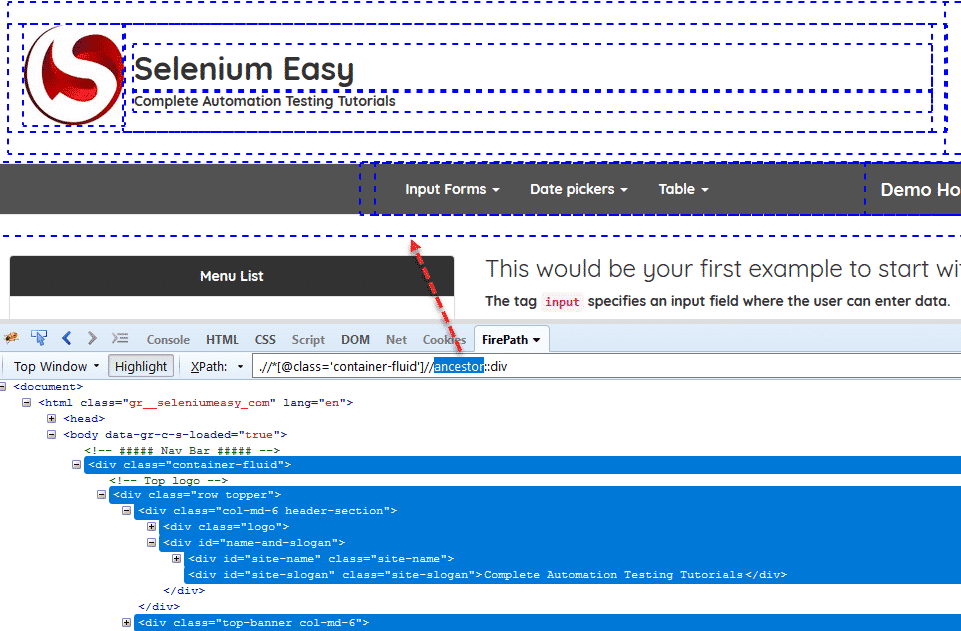
### ****Ancestor****

It finds the element before the ancestor statement and set it as a top node and then starts to **find the elements in that node**. In below example,

1- First, it finds the class which id is “container-fluid”

2- Then, starts to find div elements in that node.

**Example**: //\*[@class=’container-fluid’]//ancestor::div

[](https://224926-685269-raikfcquaxqncofqfm.stackpathdns.com/wp-content/uploads/2017/09/xpath-ancestor.png)

You can select specific div groups by changing div depths as shown below.

.//\*[@class=’container-fluid’]//ancestor::div[1] – Returns 13 nodes  
.//\*[@class=’container-fluid’]//ancestor::div[2] – Returns 7 nodes  
.//\*[@class=’container-fluid’]//ancestor::div[3] – Returns 5 nodes  
.//\*[@class=’container-fluid’]//ancestor::div[4] – Returns 3 nodes  
.//\*[@class=’container-fluid’]//ancestor::div[5] – Returns 1 node

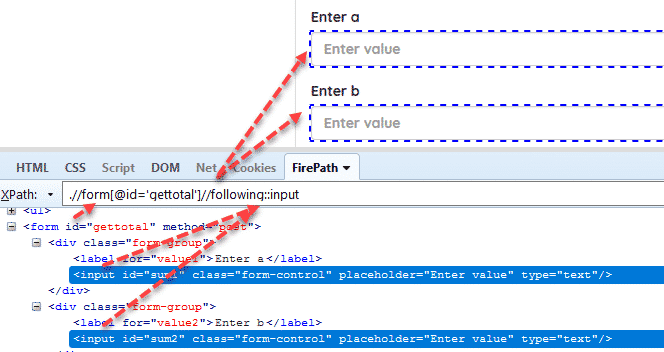
### ****Following****

Starts to locate elements **after the given parent node**. It finds the element before the following statement and set as the top node and then starts to find **all elements after that node**. In below example,

1- First, it finds the form which id is “gettotal”

2- Then, starts to find all input elements after that node.

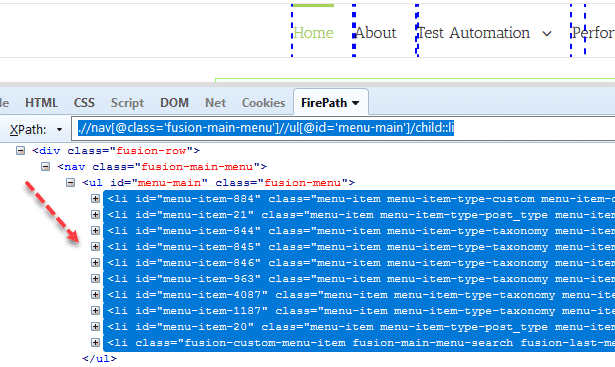
**Example**: .//form[@id=’gettotal’]//following::input

[](https://224926-685269-raikfcquaxqncofqfm.stackpathdns.com/wp-content/uploads/2017/09/xpath-following.png)

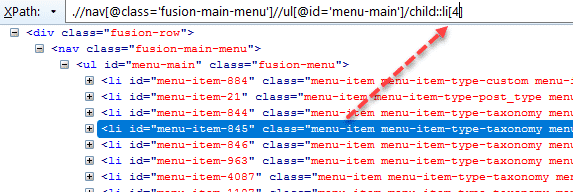
### ****Child****

Selects all children elements of the current node.

**Example**: //nav[@class=’fusion-main-menu’]//ul[@id=’menu-main’]/child::li

[](https://224926-685269-raikfcquaxqncofqfm.stackpathdns.com/wp-content/uploads/2017/09/xpath-child.png)

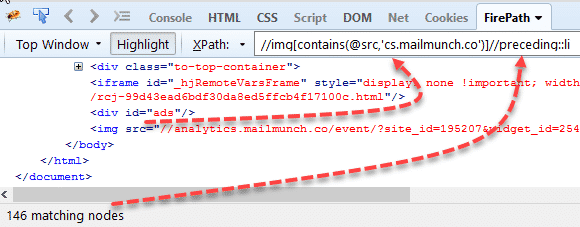
You can also **select the required “li” element by using li[1], li[2], li[3]**, etc. syntax as shown below.

[](https://224926-685269-raikfcquaxqncofqfm.stackpathdns.com/wp-content/uploads/2017/09/child-locator-tactic.png)

### ****Preceding****

Select all nodes that come before the current node. I give an example on swtestacademy. We will find all “li” elements in the homepage. First, we will locate the bottom element, then use preceding with “li” to find all “li” elements as shown below.

**Example**: //img[contains(@src,’cs.mailmunch.co’)]//preceding::li

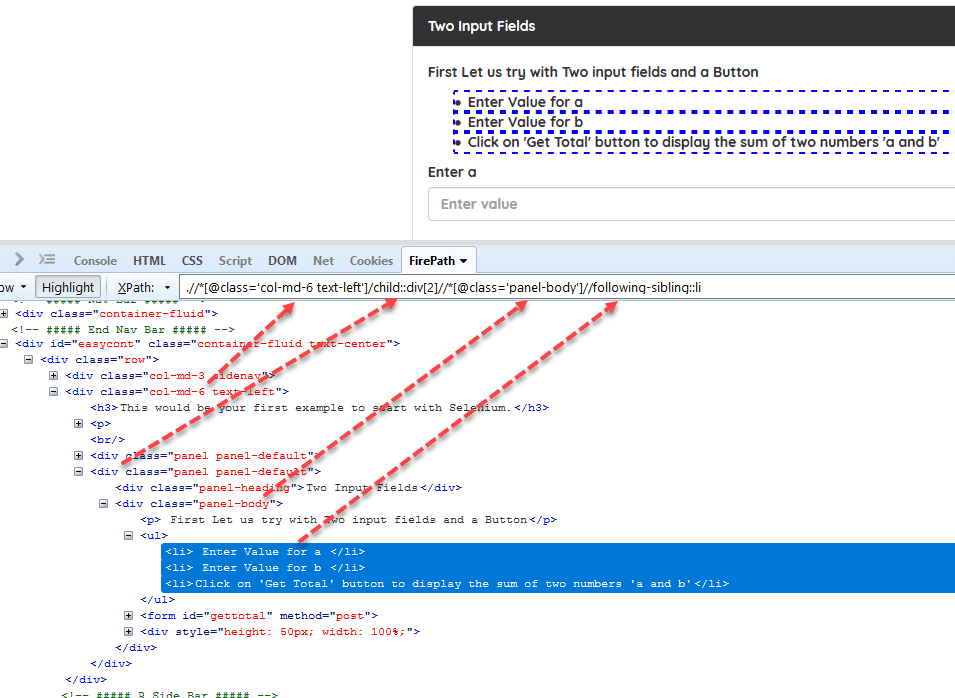
[](https://224926-685269-raikfcquaxqncofqfm.stackpathdns.com/wp-content/uploads/2017/09/preceding-locator-tactic.png)

Also, you can use [1], [2], etc. to select a specific element in the preceding element list.

### ****Following-sibling****

Select the following siblings of the context node.

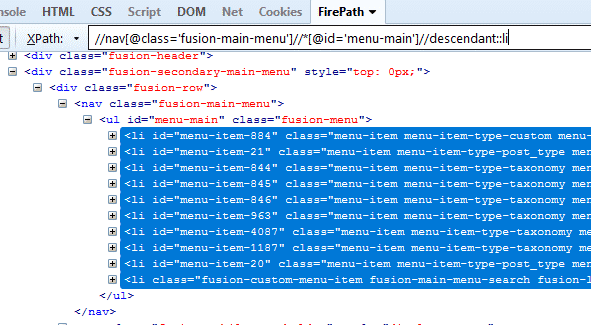
**Example**: //\*[@class=’col-md-6 text-left’]/child::div[2]//\*[@class=’panel-body’]//following-sibling::li

[](https://224926-685269-raikfcquaxqncofqfm.stackpathdns.com/wp-content/uploads/2017/09/following-sibling.png)

### ****Descendant****

Identifies and returns all the element descendants to current element which means traverse down under the current element’s node. Below, the XPath returns all “li” elements under the “menu-main”.

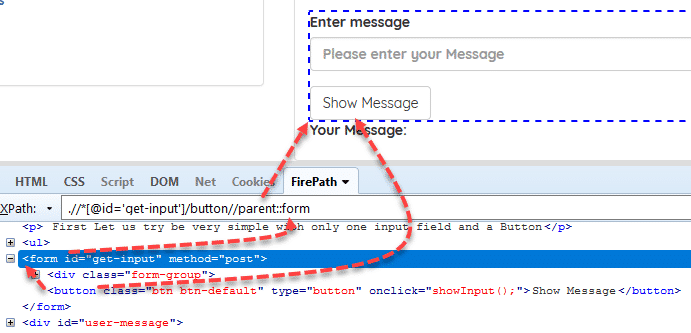
**Example**: //nav[@class=’fusion-main-menu’]//\*[@id=’menu-main’]//descendant::li

[](https://224926-685269-raikfcquaxqncofqfm.stackpathdns.com/wp-content/uploads/2017/09/descendant.png)

**Parent**

Returns the parent of the current node as shown in the below example.

**Example**: .//\*[@id=’get-input’]/button//parent::form

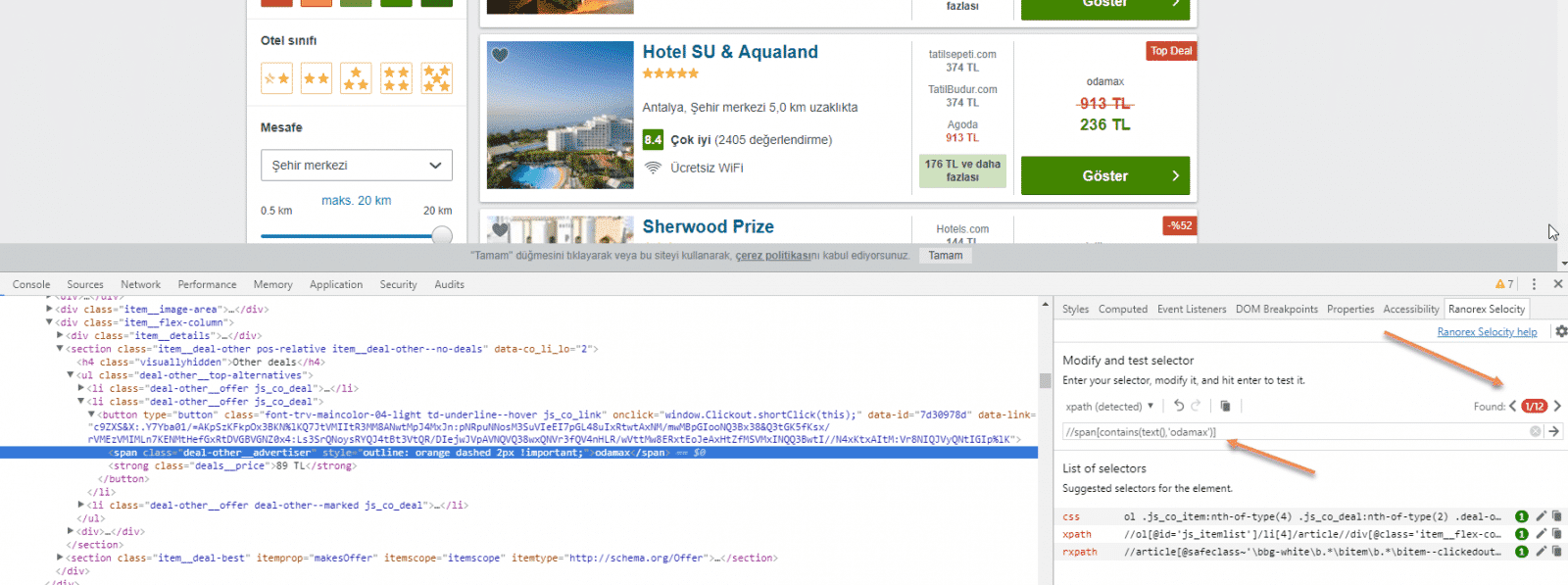
[](https://224926-685269-raikfcquaxqncofqfm.stackpathdns.com/wp-content/uploads/2017/09/parent-selector-tactic-in-selenium.png)

### ****Locate an Element inside Array of Elements****

In Trivago website, lets search “Antalya” keyword. Then, find the first Odamax hotel with XPath.

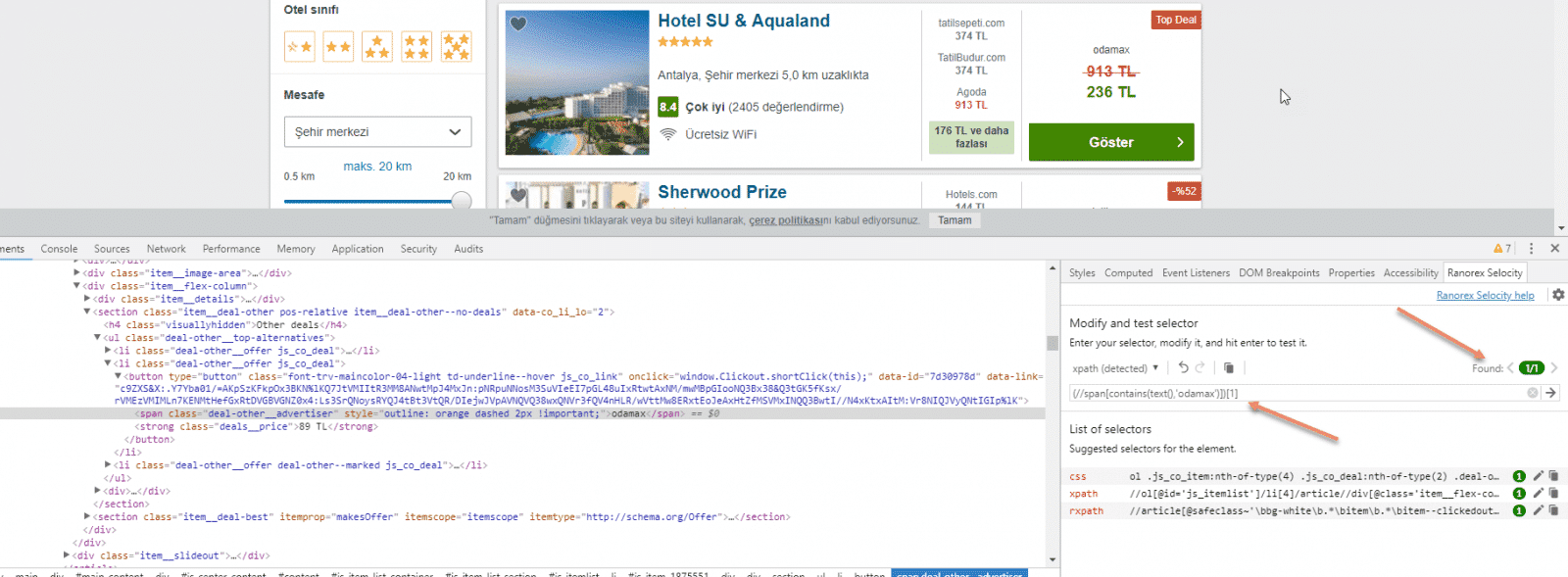
First, we can find all Odamax hotels by using its text with below XPath:

**//span[contains(text(),’odamax’)]**

[](https://224926-685269-raikfcquaxqncofqfm.stackpathdns.com/wp-content/uploads/2018/03/mutliple-element-selection-selenium.png)

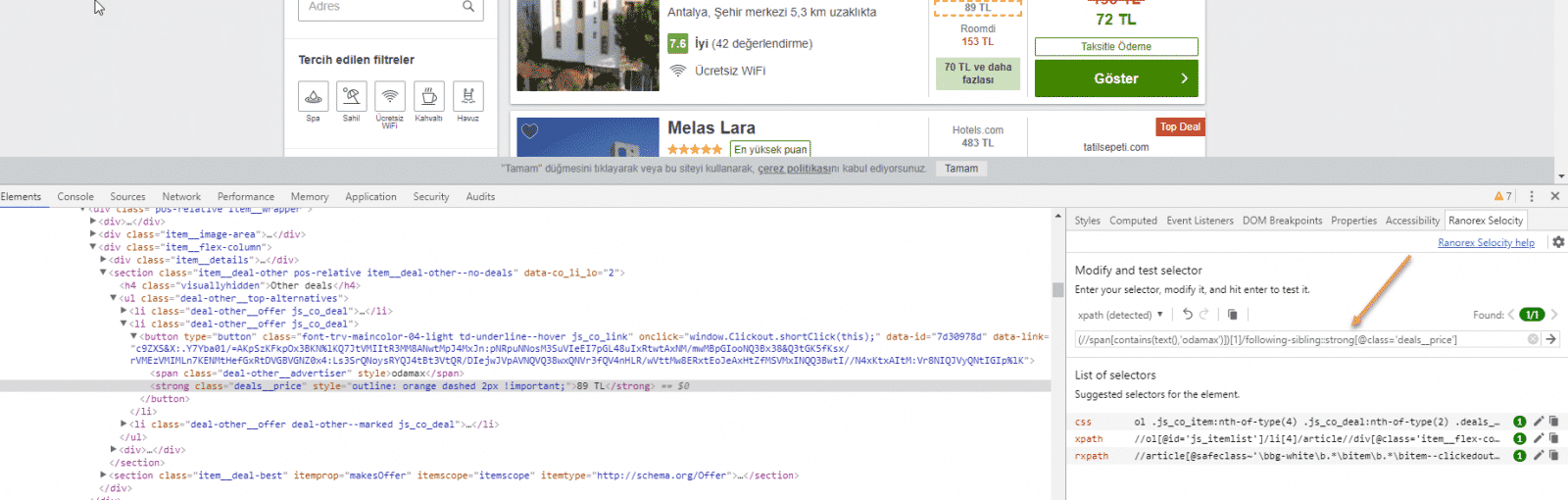
Above XPath returns many Odamax hotel’s, we can select the first one with below XPath expression:

**(//span[contains(text(),’odamax’)])[1]**

[](https://224926-685269-raikfcquaxqncofqfm.stackpathdns.com/wp-content/uploads/2018/03/xpath-array-of-elements.png)

You can also continue to search and find the related hotel’s price element with below XPath:

**(//span[contains(text(),’odamax’)])[1]/following-sibling::strong[@class=’deals\_\_price’]**

[](https://224926-685269-raikfcquaxqncofqfm.stackpathdns.com/wp-content/uploads/2018/03/complex-xpath-in-selenium.png)

**Also, you can learn how to write effective Selenium CSS locators in below article.**

## **CSS Selenium Selector Strategies**

***Reference Demo Site:****http://www.seleniumeasy.com/test/basic-first-form-demo.html*

### ****Basic Syntax****

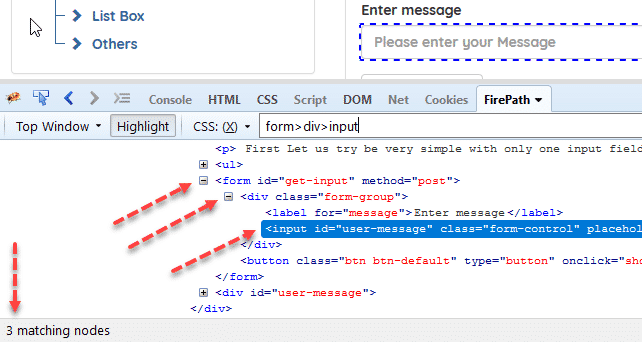
|  |  |
| --- | --- |
| **ID** | **#idname** |
| **Class** | **.classname** |

### ****1) Using Absolute Path****

You should **use > sign** to reach an element directly.  (Note: For XPath we use***/ sign*** for this.)

**Example**

**Syntax**: **form>div>input**

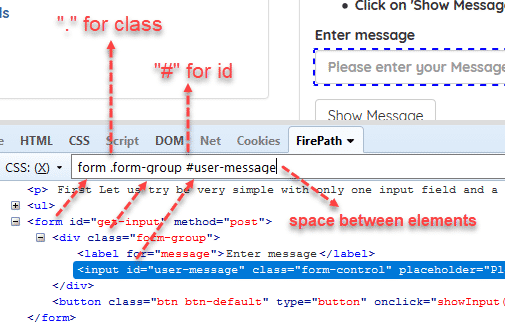
[](https://224926-685269-raikfcquaxqncofqfm.stackpathdns.com/wp-content/uploads/2017/09/img_59cee3e5d9728.png)

### ****2) Using Non-Absolute Path****

You should use a **white space** between tags to locate the element. Use **“.” for class** and **“#” for id**.

**Example**

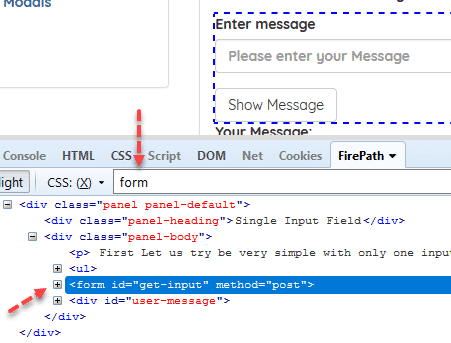
**Syntax**: **form .form-group #user-message**



### ****3) Using Tag Name****

You can **write the tag name directly** like “form”, “div”, “img”,”body”, etc. As below figure, I wrote “form” tag to locate the form element.  
(Note: For XPath we use ***//tagname*** for this.)

**Example**

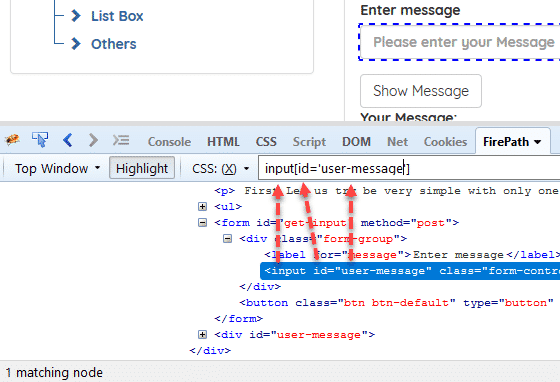


### ****4) Using Tag & Attribute & Value Trio****

You can use **tag[attribute=’value’]** syntax.  
(Note: For XPath we use ***tag[@attribute=’value’]*** for this.)

**Example**

**Syntax**: **input[id=’user-message’]**

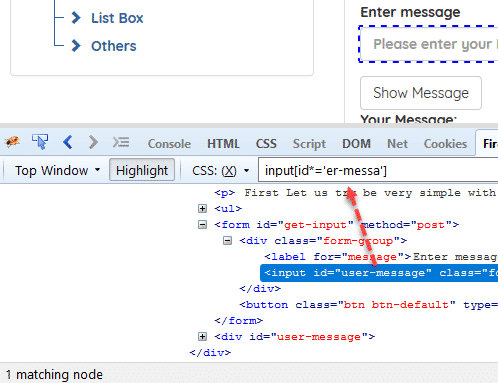


### ****5) Using Containing Text of an Attribute****

You can use **tag[attribute\*=’containing text’]** syntax.  
(Note: For XPath we use ***tag[contains((@attribute,’containing text’)]*** for this.)

**Example**

**Syntax**: **input[id\*=’er-messa’]**

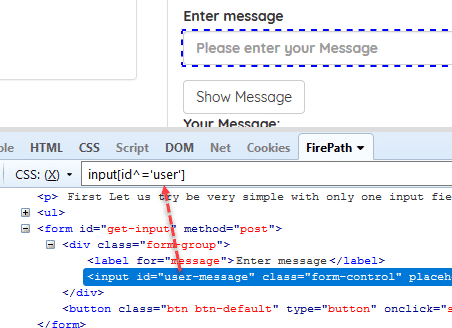
[](https://224926-685269-raikfcquaxqncofqfm.stackpathdns.com/wp-content/uploads/2017/09/img_59cef158ea5c9.png)

### ****6) Using Starting Text of an Attribute****

You can use **tag[attribute^=’starting text’]** syntax.  
(Note: For XPath we use  ***tag[starts-with( @attribute, ‘starting text’)]*** for this.)

**Example**

**Syntax**: **input[id^=’user’]**

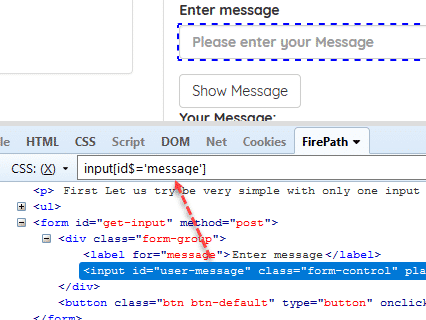
[](https://224926-685269-raikfcquaxqncofqfm.stackpathdns.com/wp-content/uploads/2017/09/img_59cef24a58d7d.png)

### ****7) Using Ending Text of an Attribute****

You can use **tag[attribute$=’ending text’]** syntax.

**Example**

**Syntax**: **input[id$=’message’]**

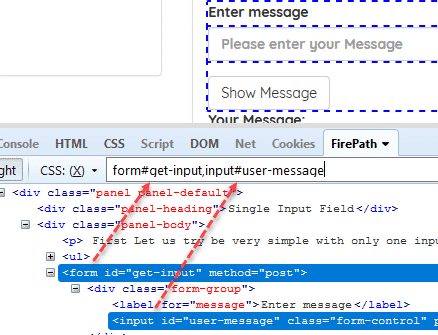
[](https://224926-685269-raikfcquaxqncofqfm.stackpathdns.com/wp-content/uploads/2017/09/img_59cef35c9f52b.png)

### ****8) Using Comma Operator to Implement OR Operation****

You can use **“,” operator between two CSS locator statements**.

**Example**

**Syntax**: **form#get-input,input#user-message**

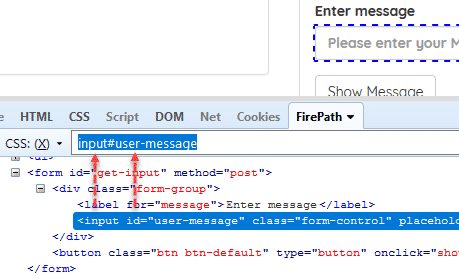
[](https://224926-685269-raikfcquaxqncofqfm.stackpathdns.com/wp-content/uploads/2017/09/img_59cef8d647a27.png)

### ****9) Using Tag and ID****

You can use **“Tag#Id”**

**Example**

**Syntax**: **input#user-message**

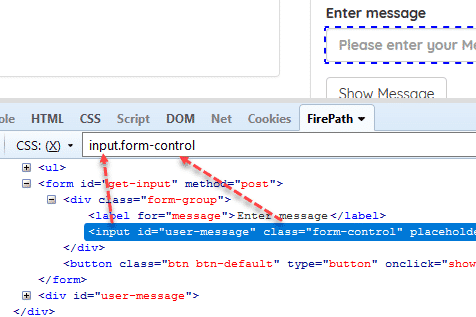
[](https://224926-685269-raikfcquaxqncofqfm.stackpathdns.com/wp-content/uploads/2017/09/img_59cf807eef983.png)

### ****10) Using Tag and Class****

You can use **“Tag.Class”**

**Example**

**Syntax**: **input.form-control**

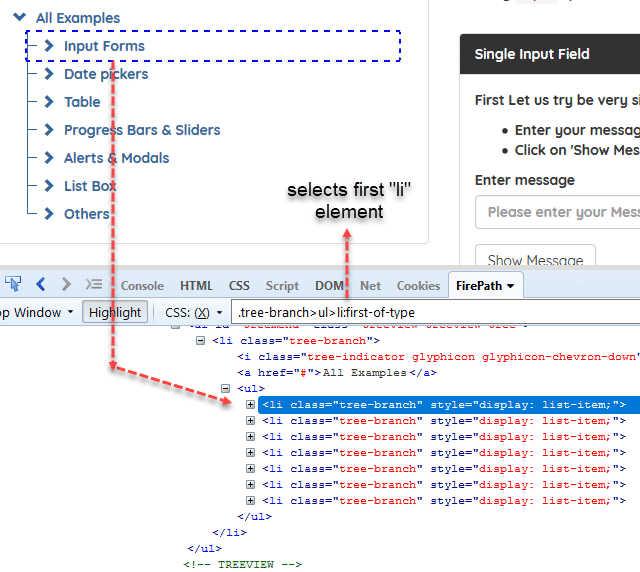
[](https://224926-685269-raikfcquaxqncofqfm.stackpathdns.com/wp-content/uploads/2017/09/img_59cf811b3166d.png)

### ****11) Using first-of-type****

You can use **“Tag:first-of-type”**. It will select the first tag element.

**Example**

**Syntax**: **.tree-branch>ul>li:first-of-type**

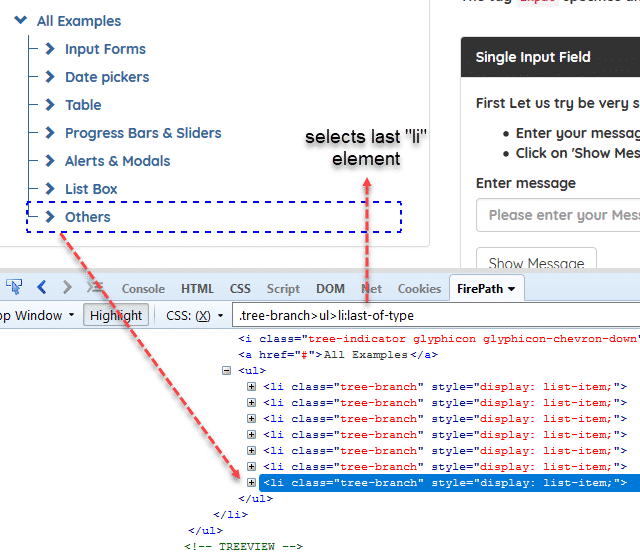
[](https://224926-685269-raikfcquaxqncofqfm.stackpathdns.com/wp-content/uploads/2017/09/img_59cf8803abe2d.png)

### ****12) Using last-of-type****

You can use **“Tag:last-of-type”**. It will select the last tag element.

**Example**

**Syntax**: **.tree-branch>ul>li:first-of-type**

[](https://224926-685269-raikfcquaxqncofqfm.stackpathdns.com/wp-content/uploads/2017/09/img_59cf903e31130.png)

**Note: If you want to find last element or child you can use below locators.**

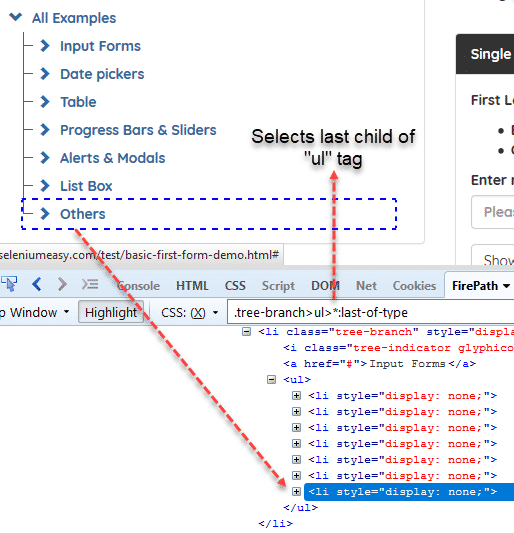
* **Tag:nth-last-of-type(n)**
* **Tag:nth-last-child(n)**

### ****13) Using \*:last-of-type****

You can use **“\*last-of-type”**. It will select the last child of parent tag.

**Example**

**Syntax**: **.tree-branch>ul>\*:last-of-type** (Selects the last child of parent tag “ul”.)

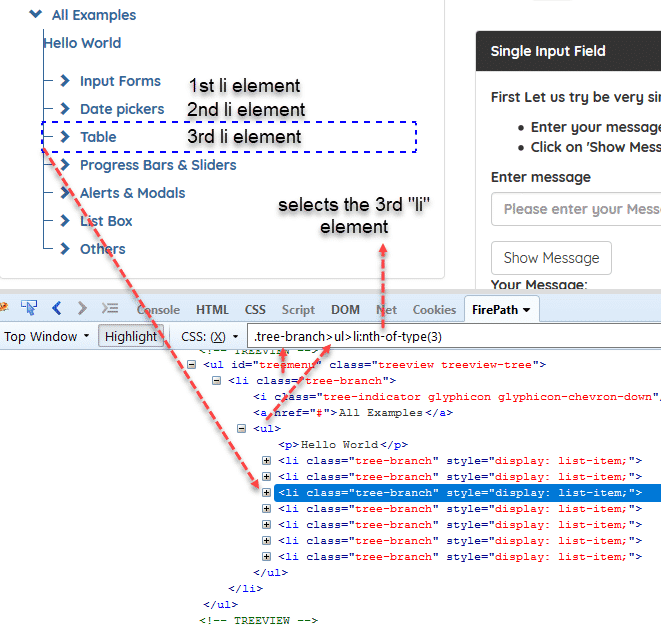
[](https://224926-685269-raikfcquaxqncofqfm.stackpathdns.com/wp-content/uploads/2017/09/img_59cf91e4f2bfb.png)

### ****14) Using tag:nth-of-type(****n****)****

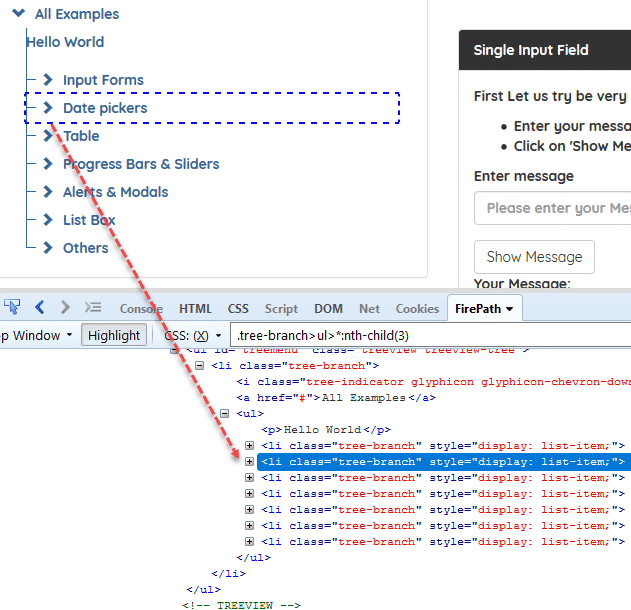
You can use **“tag:nth-of-type(n)”**. It will select the nth tag element of the list.

**Example**

**Syntax**: **.tree-branch>ul>li:nth-of-type(3)**(Selects 3rd li element.)

[](https://224926-685269-raikfcquaxqncofqfm.stackpathdns.com/wp-content/uploads/2017/10/img_59d009e1112d6.png)

**Note:** If you don’t specify a tag as  **\*:nth-of-type(3)** it will allow you to **select the third child**.

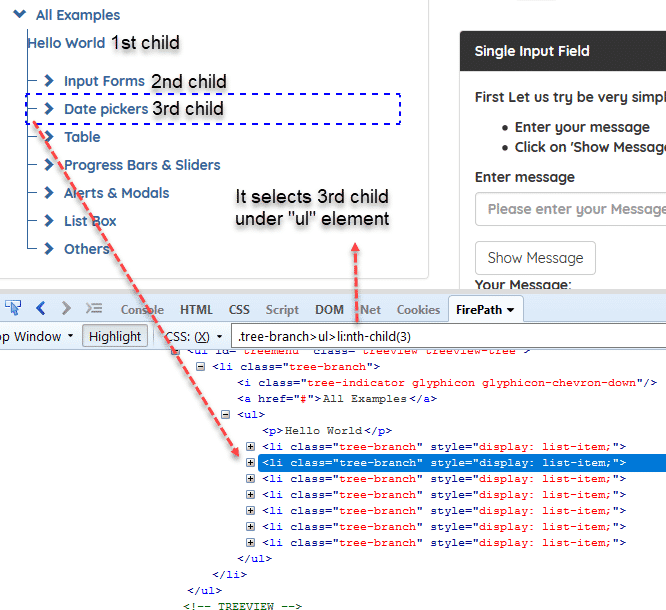
[](https://224926-685269-raikfcquaxqncofqfm.stackpathdns.com/wp-content/uploads/2017/10/img_59d00a348724e.png)

### ****15) Using tag:nth-child(****n****)****

You can use **“tag:nth-child(n)”**. It will select the nth child.

**Example**

**Syntax**: **.tree-branch>ul>li:nth-child(3)**(It will select the nth child.)

[](https://224926-685269-raikfcquaxqncofqfm.stackpathdns.com/wp-content/uploads/2017/10/img_59d009899217e.png)

### ****16) Using Sibling “+” Operator****

You can use **“E1+ E2“**. First, it finds E1 then select E2.

**Sample HTML:**

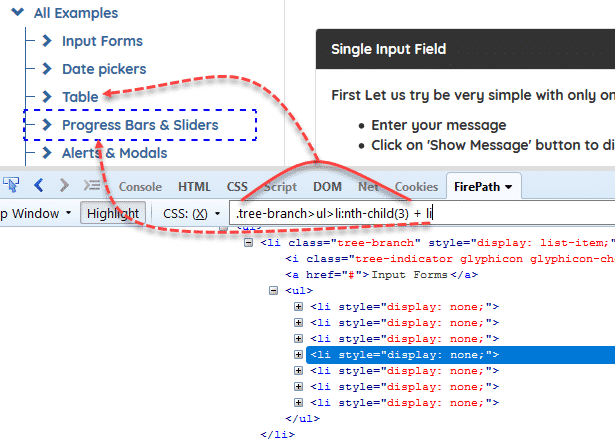
|  |  |
| --- | --- |
| 1  2  3  4  5 | <ul id="Cars">     <li id="mercedes">Mercedes made in Germany!</li>     <li>BMW</li>     <li>Porsche</li>  </ul> |

**Syntax: li#mercedes + li**

‘**li#automation + li**‘ will first go to li element with id ‘mercedes’ and **then select its adjacent li** which is ‘BMW’ list item.

**Example**

**Syntax**: **.tree-branch>ul>li:nth-child(3) + li**(It will select the next element.)

[](https://224926-685269-raikfcquaxqncofqfm.stackpathdns.com/wp-content/uploads/2017/10/img_59d01422609b9.png)

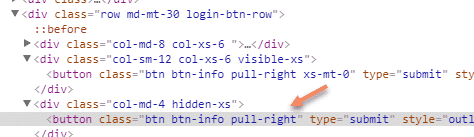
### ****17) Exclude a CSS Class Name in CSS Selenium Selector****

You can exclude any of the class name with **:not(.class-name)** syntax.

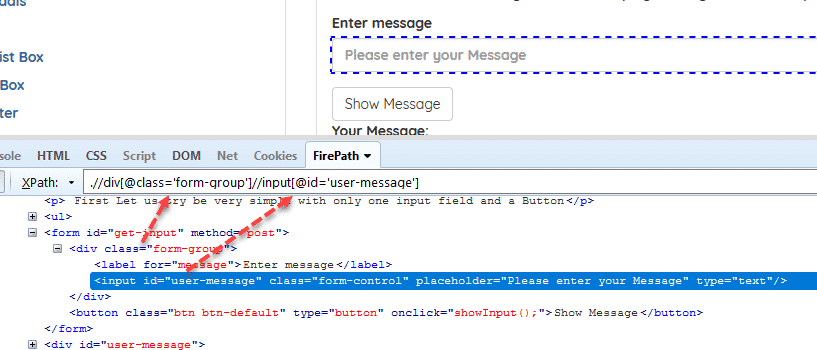
**Example:**

**.btn.btn-info.pull-right:not(.xs-mt-0)**

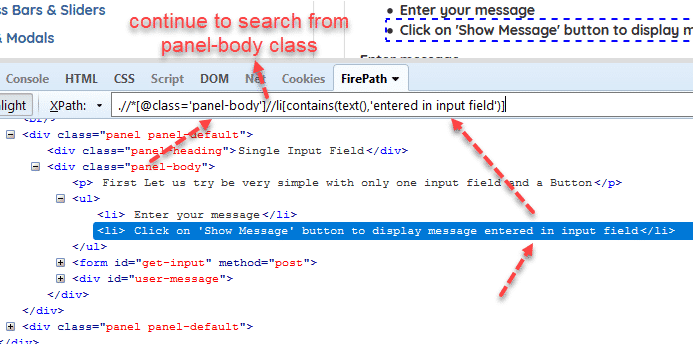
Above selector excludes “**xs-mt-o**” class and selects below line as shown below figure.

[](https://224926-685269-raikfcquaxqncofqfm.stackpathdns.com/wp-content/uploads/2018/03/img_5aa16cdd1ccea.png)

**Also, you can learn how to write Smart XPath locators in below**

[](https://224926-685269-raikfcquaxqncofqfm.stackpathdns.com/wp-content/uploads/2017/09/relative-xpath.png)

**Example:**//\*[@class=’panel-body’]**//**li[contains(text(),’entered in input field’)]

[](https://224926-685269-raikfcquaxqncofqfm.stackpathdns.com/wp-content/uploads/2017/09/relative-xpath-2.png)