Table of Contents

[I. **X-PATH** 2](#_Toc12610541)

[**1.** **Types of X-Path** 3](#_Toc12610542)

[a. Absolute XPath 3](#_Toc12610543)

[b. Relative XPath 4](#_Toc12610544)

[**2.** **Function in XPath** 4](#_Toc12610545)

[a. Or & And 5](#_Toc12610546)

[b. Text, Contains & Starts-with/Ends-With 6](#_Toc12610548)

[c. Function using XPath-Axis 9](#_Toc12610549)

[**3.** **How to get XPath effectively** 11](#_Toc12610550)

[a. Tab and list 11](#_Toc12610551)

[b. Tab inside Tab 12](#_Toc12610552)

[4. **Conclusion** 13](#_Toc12610553)

[II. **CSS SELECTORS** 13](#_Toc12610554)

[1. 13](#_Toc12610555)

[III. **CONCLUSION AND CAREFUL** 14](#_Toc12610556)

[**1. Compare XPath and CSS Selectors** 14](#_Toc12610557)

[**2. XPath and CSS selectors notes** 16](#_Toc12610558)

**XPATH AND CSS**

# X-PATH

XPath is a syntax for defining parts of an XML document. XPath uses path expressions to select nodes or node-sets in an XML document. These path expressions look very much like the expressions you see when you work with a traditional computer file system.

XPath expressions can be used in JavaScript, Java, XML Schema, PHP, Python, C and C++, and lots of other languages. The structure an XML document is as follows:

<?xml version="1.0" encoding="UTF-8"?>  
<bookstore>  
  <book>  
    <title lang="en">Harry Potter</title>  
    <author>JK. Rowling</author>  
   <year>2005</year>  
    <price>29.99</price>  
  </book>  
</bookstore>

Example of nodes in the XML document above:

<bookstore> (root node)  
  
<author>J K. Rowling</author> (node with text)  
  
<title lang="en">Harry Potter</title> (node with attribute)

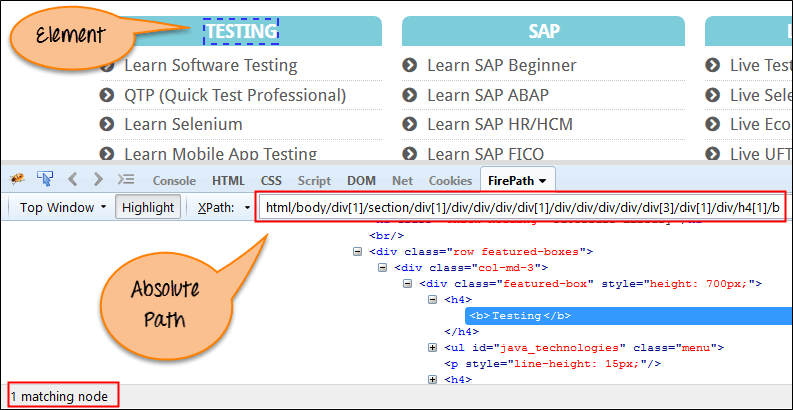
Note that a button includes Tag Name, may contain Text Content and one or more Attribute

## Types of X-Path

There are 2 XPath types categorized by the structure of the path expression: Absolute XPath and Relative XPath

### Absolute XPath

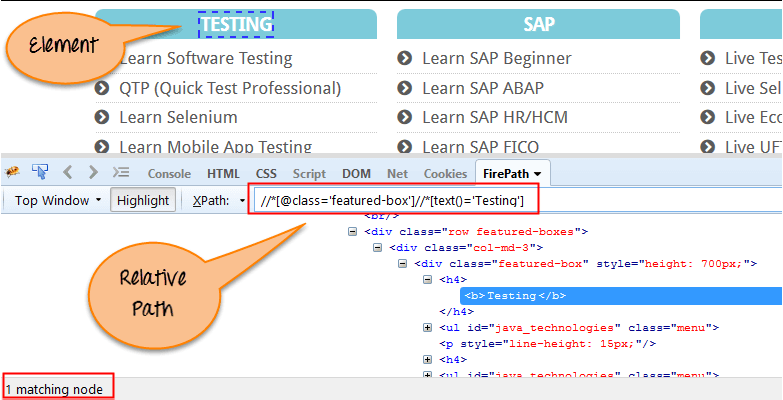
It is the direct way to find the element, but the disadvantage of the absolute XPath is that if there are any changes made in the path of the element then that XPath gets failed. The key characteristic of XPath is that it begins with the single forward slash (/), which means you can select the element from the root node.

Below is the example of an absolute XPath expression of the element shown in the below screen.

html/body/div[1]/section/div[1]/div/div/div/div[1]/div/div/div/div/div[3]/div[1]/div/h4[1]/b

### Relative XPath

For Relative XPath the path starts from anywhere of the XML DOM structure. It starts with the double forward slash (//), which means it can search the element anywhere of XML DOM. Combined with the attribute and text of the element to determine the location.

Below is the example of a relative XPath expression of the same element shown in the below screen. This is the common format used to find element through a relative XPath.

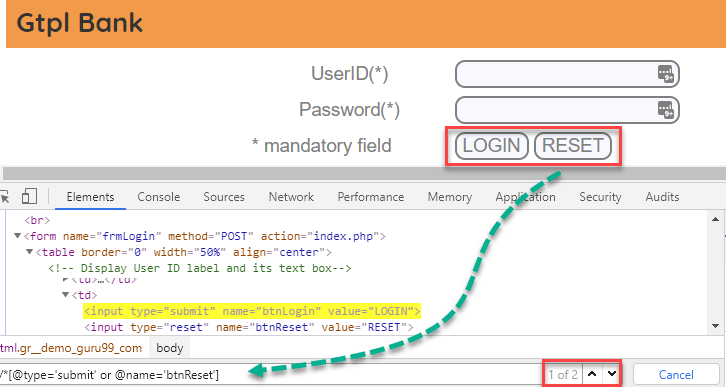
Relative xpath: //\*[@class='featured-box']//\*[text()='Testing']

## Function in XPath

XPath includes over 200 built-in functions. The contents of this document only mention the most used and effective functions: or & and, text, contains, starts-with/ends-with and functions using XPath axis.

### Or & And

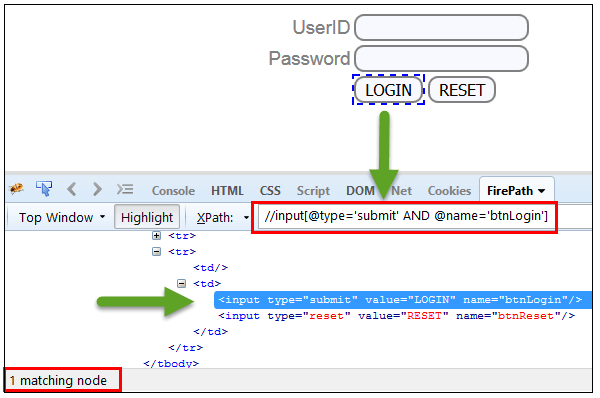
In OR expression, two conditions are used, whether 1st condition OR 2nd condition should be true. It is also applicable if any one condition is true or maybe both. Means any one condition should be true to find the element.

In the below XPath expression, it identifies the elements whose single or both conditions are true. Highlighting both elements as "LOGIN" element having attribute 'type' and "RESET" element having attribute 'name'.

Xpath=//\*[@type='submit' or @name='btnReset']

In AND expression, two conditions are used, both conditions should be true to find the element. It fails to find element if any one condition is false.

I**n below expression, highlighting 'LOGIN' element as it having both attribute 'type' and 'name'.**

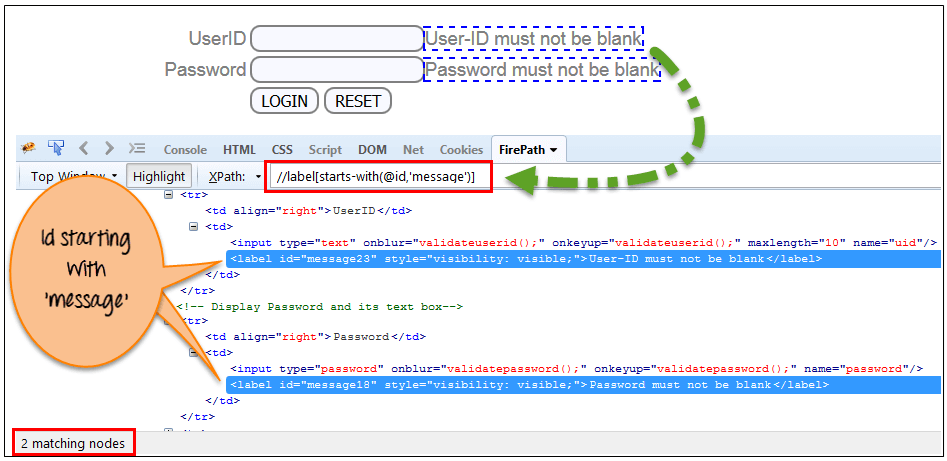


Xpath=//input[@type='submit' and @name='btnLogin']

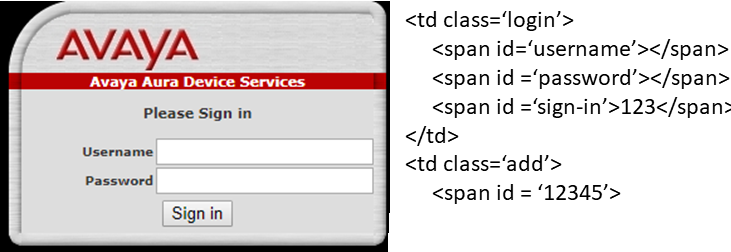
### Text, Contains & Starts-with/Ends-With

Starts-with( method is used when we know about the initial partial attribute value or initial partial text associated with the element. User can also use this method to locate elements those are consist of both the static(initial) and dynamic(trailing) values. In contrast Starts-with is Ends-with using the end values ​​of the attribute or the end of the text to determine the node.

In below example, XPath finds those element whose 'ID' starting with 'message'.

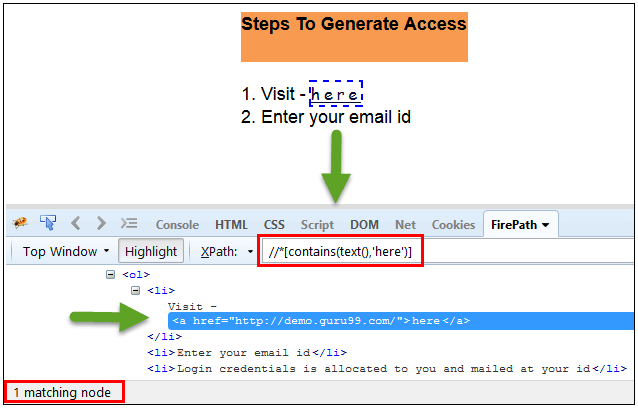


Xpath=//label[starts-with(@id,'message')]

In this expression, with text function, we find the element with exact text match as shown below. In our case, we find the element with text "".

Xpath=//td[text()='UserID'

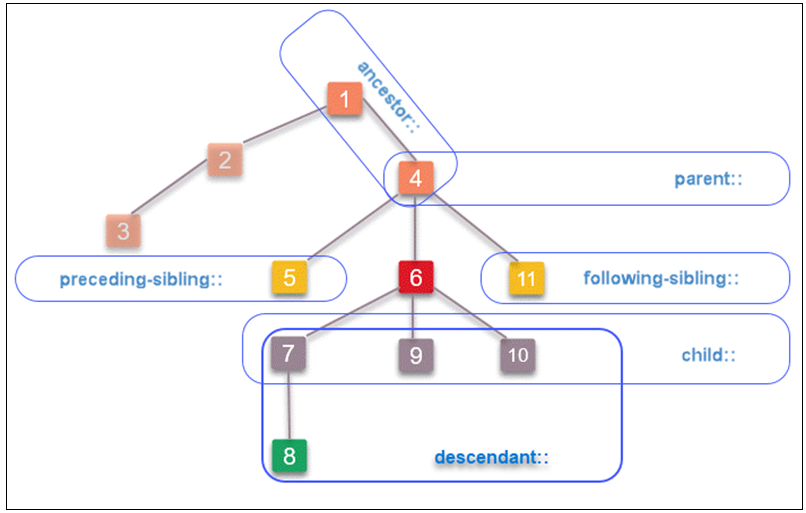
Contains method is used when we know about the partial attribute value or partial text associated with the element.

In the below expression, we have taken the "text" of the link as an attribute and 'here' as a partial value as shown in the below screenshot. This will find the link ('here') as it displays the text 'here'.

Xpath=//\*[contains(text(),'here')]

Xpath=//\*[contains(@href,'guru99.com')]

### Function using XPath-Axis

In order to understand the functions using XPath-axis, we need to understand the relationships of the nodes in the xml document. At a node defined in the XML document it will include the following relationships: Parent, child, ancestor, sibling and descendant. The image below illustrates the possible relationships of the node

Node [6] with relationships: ancestor, parent, sibling, chill, descendant

Corresponding to the above relationships, we have the corresponding XPath functions to easily locate the relevant elements from a known node.

Based on the relationship image of the above nodes, we have an example of the corresponding XML document below

<?xml version="1.0" encoding="UTF-8"?>  
<1 name="This is ancestor of [6]">  
   <2 name="node none has relationship with [6]">

<3 name="node none has relationship with [6]"></3>

</2>

<4 name="This is parent and ancestor of [6]">

<5 name="This is elder brother of [6]"></5>

<6 name="This is node [6]" id="6">

<7 name="This is child and descendant of [6]"></7>

<8 name="This is descendant of [6]"></8>

<9 name="This is child and descendant of [6]"></9>

<10 name="This is child and descendant of [6]"></10>

<11 name="This is younger brother of [6]"></11>

</6>

</4>  
</1>

Node [6] //\*[@id="6"] -> [6]

ancestor: //\*[@id="6"]/ancestor::\* -> [4][1]

parent: //\*[@id="6"]/parent::\* -> [4]

sibling: //\*[@id="6"]/preceding-sibling::\* -> [5]

sibling: //\*[@id="6"]/following-sibling::\* -> [11]

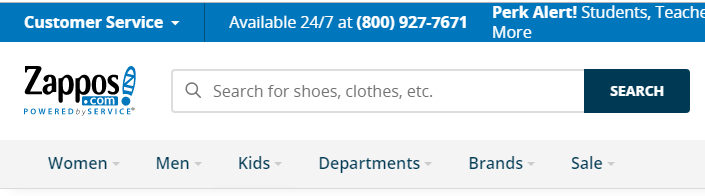
child: //\*[@id="6"]/child::\* -> [7][9][10]

descendant: //\*[@id="6"]/descendant::\* -> [7][8][9][10]

so when XPath axis functions are used. In fact, XPath links can’t always be determined based on the basic XPath functions. One of them is that there are many elements in the XML file with the same properties. Or in case the elements are dynamic elements that always change the price when we manipulate the XML file. At the same time, the XPath axis method is also applied in writing general locators for many elements with similarities

## How to get XPath effectively

### Tab and list

For cases where the element is a list or the Tab has the same relationship with a node that determines, they are different only in the text content or the value of the same property, then we can apply the method. Write a common locator for these locations. Here is an illustrative example with the element is a tab that content text different and is descendant node of <tab name = 'tab'>

<?xml version="1.0" encoding="UTF-8"?>  
<Zappos id ="" name="">  
   <tab name="tab">

<ul name="">

<span id="">Women</span>

</ul>

<ul name="">

<span id="">Men</span>

</ul>

<ul name=">

<span id="">Kids</span>

</ul>

.

.

.

</tab>

</Zappos>

By.xpath("//tab[@name=’tab’]//span[contains(text(),'"+name+"')]";

From there we just created a function in Scripts to return XPath with a “String name” variable. Do the same for a list element

### Tab inside Tab

Problem: based on the general Locator method for cases with multiple Tab and each Tab has multiple Sub Tab. To solve it, we use two variables corresponding to the names of Main Tab and Sub Tab. An example for this case is described below. Example, include the "Women" and "Men" Tab both contain 2 Subtab as "heels" and "sneakers".

<?xml version="1.0" encoding="UTF-8"?>  
<Zappos id ="" name="">  
   <tab name="tab">

<span id="Women">

<a id="">heels</a>

<a id="">sneakers</a>

</span>

<span id="">Men</span

<a id="">heels</a>

<a id="">sneakers</a>

.

.

.

</tab>

</Zappos>

By.xpath(“//\*[@name=’tab’]//span[@id]=’”+tab+”’]/following-sibling::a[@id=’subtab’]”)

## Conclusion

XPath is required to find an element on the web page as to do an operation on that particular element.

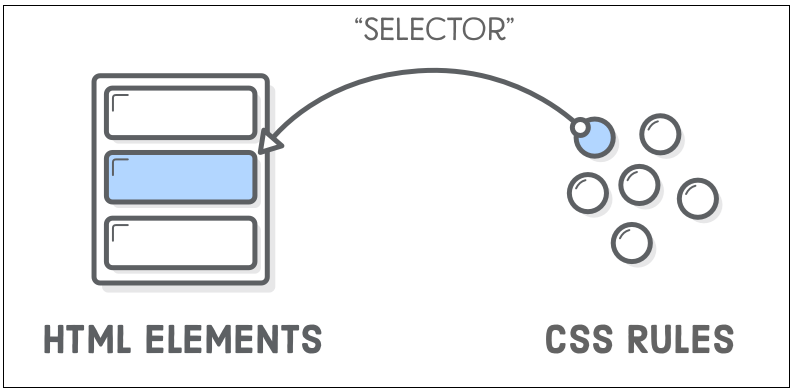
There are two types of XPath:

* Absolute XPath
* Relative XPath

XPath Axes are the methods used to find dynamic elements, which otherwise not possible to find by normal XPath method

XPath expression select nodes or list of nodes on the basis of attributes like ID, Name, Class, etc. from the XML document.

# CSS SELECTORS

Similar to XPath, CSS is the selector that looks for elements in the HTML document.

CSS Selector also uses attributes like ID, Class, Tagname ... to determine the path for the desired element.

## Function of CSS Selectors

### ID, class, attribute, and

As mentioned in the previous section. CSS Selector also uses attributes like ID, Class, Tagname ... and operators to define the path for the desired element. Below is an example of the basic functions in the CSS selector.

<?xml version="1.0" encoding="UTF-8"?>  
<bookstore>  
  <book>  
    <title class="abc">Harry Potter</title>  
    <author id="123" class="xyz">JK. Rowling</author>  
    <year id="123" class="xyz">2005</year>  
    <price style="def">29.99</price>  
  </book>  
</bookstore>

<author> using Class and ID: author.xyz#123

<year> using ID: year#123

<price> using value attribute: price[style=’def’]

### Contains, child, descendant, next-element

Similar to XPath, CSS Selector also using functions: Contains, Child, Descendant. However, CSS cannot use the contains function to specify content outside of the Attributes of Node and it has no text() function. One of the important is CSS only works in the transition direction. This means that we can not get locate parent, ancestor elements with child/descendant elements. Below is an example of contains, child, descendant, next-element functions in the CSS selector.

<?xml version="1.0" encoding="UTF-8"?>  
<1 name="This is ancestor of [6]">  
   <2 name="node none has relationship with [6]">

<3 name="node none has relationship with [6]"></3>

</2>

<4 name="This is parent and ancestor of [6]">

<5 name="This is elder brother of [6]"></5>

<6 name="This is node [6]" id="6">

<7 name="This is child and descendant of [6]"></7>

<8 class="abc" name="This is descendant of [6]"></8>

<9 name="This is child and descendant of [6]"></9>

<10 name="This is child and descendant of [6]"></10>

<11 name="This is younger brother of [6]"></11>

</6>

</4>  
</1>

<6>: 6#6 -> [6]

Child 6#6 >\* -> [7][9][10]

Fisrt-child 6#6 >\*:first-child -> [7]

N child 6#6 >\*:nth-child(3) -> [10]

Descendant 6#6 .abc -> [8]

Next-element 6#6 -> [7]

## Conclusion

CSS has a simple structure, High performance speed. Works well on the Browser (especially in IE).

# CONCLUSION AND CAREFUL

## Compare XPath and CSS Selectors

|  |  |  |
| --- | --- | --- |
| Goal | CSS | XPath |
| All element | \* | //\* |
| All P element | P | //p |
| All child element | P >\* | //p/\* or //p//child::\* |
| Element by ID | #foo | //\*[@id=’foo’] |
| Element by Class | .foo | //\*[@class=’foo’] |
| Element with Attribute | \*[title] | //\*[@title] |
| First child of all p | P > \*:first-child | //p/\*[1] or //p//child::\*[1] |
| n child of all p | P:nth-child(n) | //p/\*[n] or //p//child::\*[n] |
| Next element | P +\* | //p/following-sibling::\*[1] |
| All above same level | **Not Possible** | //preceding-sibling::\* |
| All bellow same level | **Not Possible** | //following-sibling::\* |
| All attribute contains | [[attribute~=value]](https://www.w3schools.com/cssref/sel_attribute_value_contains.asp) | //\*[contains(@attribute,’value’)] |
| All descendant | .class1 .class2 | //descendant::\* |
| All ancestor | **Not Possible** | //ancestor::\* |
| All parent | **Not Possible** | /parent::\* or (p/..\*) |
| All text value y | **Not Possible** | //\*[text(),’y’] |
| All text contains y | **Not Possible** | //\*[contains(text(),’y’)] |
| All text starts-with | **Not Possible** | //\*[starts-with(text(),’value’)] |
| All one in two condition | **Not Possible** | //\*[@x='y' or @z='t'] |
| All with two condition | .class1.class2 | //\*[@x=’y’ and @z=’t’] |
| All attribute starts-with | [attribute|=value] | //\*[starts-with(@attribute,’value’)] |
| All attribute ends-with | [attribute$=value] | //\*[ends-with(@attribute,’value’)] |

Note that it is possible to combine the conditions behind the functions in the table ex: “/following-sibling**::**p” or “p/following-sibling**::**p[@x=’y’]”…

* Conclusion

CSS has a simpler structure than XPath

XPath works on general XML (including well-formatted HTML) while CSS and DOM are implemented on HTML.

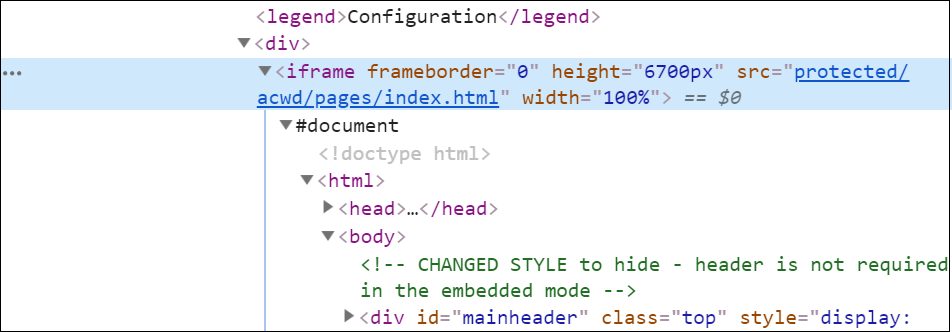
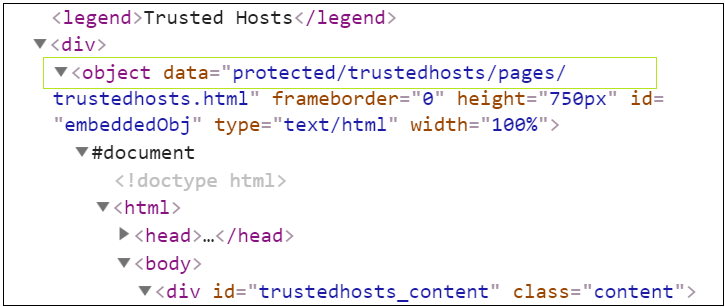
One of the important differences between XPath and CSS is that, with XPath, we can search elements back or forth in the DOM hierarchy while CSS only works in

the transition direction. This means that with XPath, we can locate parent elements with child elements.

XPath can get specify content outside of properties by Contains() or Text() while CSS is not executable

XPath of the browser is not always complete (especially in IE).

## XPath and CSS selectors notes

In the case XPath and CSS Selector defining a node is the descendant of the following special node: iframe, object, form. These special nodes are defined as sub-Web pages or sub-XML documents of the original XML document. To perform actions from the Selenium library on these respective nodes, we first need to narrow the scope of searching for XPath and CSS paths inside Web-sub / XML-sub using the swithTo () function in the Selenium library or use Javascriptexecutor to convert the search into Web-sub / XML-sub. After swithTo () to the special node to return to search for Web-sub / XML-sub external elements we use defaultContent () of Selenium or using javascriptexecutor