

# XPath

## Introduction

Selenium test automation engineers **must** be comfortable in locating elements in web pages. XPath and CSS are the most powerful location strategies used in Selenium as compared to other location strategies (id, name, className, linkText, partialLinkText and [tagName](#) )

Mastering XPath and/or CSS is essential for the Selenium test automation engineers to locate dynamic web elements, elements without ids or names and elements with dynamic ids and names. Tagname is not a useful location strategy to locate elements as there will be many elements with the same tagname. LinkText is only useful for the anchor <a> tags only. Also, it is not useful when the visible text changes as in multilingual systems

It is noted that most of the amateur Selenium automation engineers do not pay much attention to master location strategies. Recording and playback will not work with the applications with dynamic elements. This leads to failure of the automated test scripts (brittle) when web pages with dynamic contents are automated. Most of the testers rely on extracting the XPaths from browser plugins. These tools have limitations and do not provide the best XPath for dynamic elements.

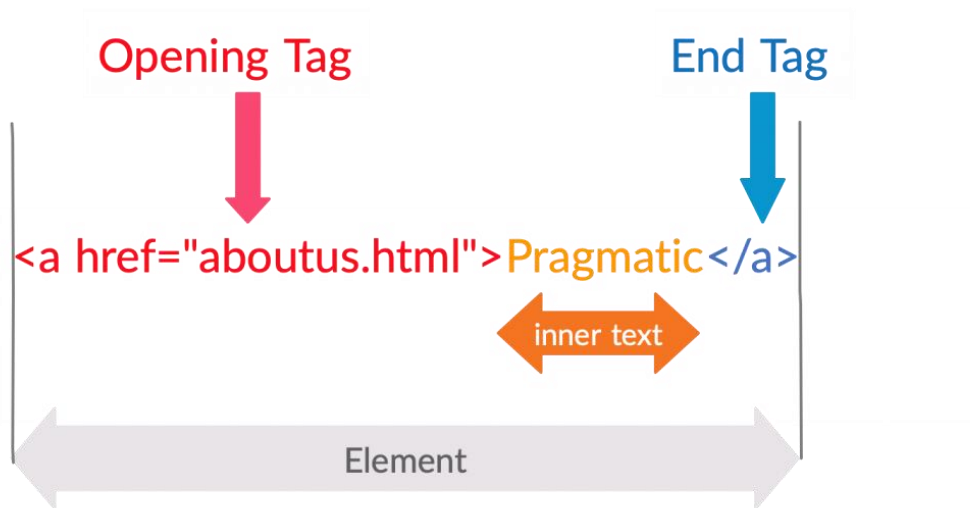
We will discuss XPath in detail with examples and explore a few tools to generate XPaths easily.

## Terms used in XPath

Lets get familiar with the basic terms used in XPath syntax.

**Tag name**

```
<input name='txtUsername' id='txtUsername' type='text'>
```



## Using XPath for locating elements in HTML

We have our own way of introducing (explaining) XPath to the trainees in our public training programs.

We write following two equations.  $A=B$  and  $B=C$ . We ask the students what can be derived from these two expressions. Immediately students reply with the answer  $A=C$ , even before the question is asked :-).

Then we give following two statements

- - XPath can be used for locating the elements in XML
  - XML and HTML have similar syntax (HTML is a subset of XML)

Hence we can derive **XPath** can be used **for locating elements in HTML** pages (web pages) too. Selenium uses XPath to locate elements in web pages.

## Why do we need to master many XPath syntaxes?

We may not be able to locate some elements using their ID or Name as some elements do not have unique attributes (id or name). Some attributes are dynamically changed. There could be elements without any attribute too. Hence we may have to locate them differently than the static elements.

XPath can be used for

- - Locating elements with respect to a known element
  - Locating elements with partially static attribute values
  - Locating elements without attributes or without unique attributes

XPath can do **bidirectional** navigation (Going forward and backward)

XPath is one of the most **flexible** and strongest location strategies.

## Types of XPath

We have grouped XPaths into 2 types.

### Absolute XPath

Absolute XPaths starts with the root of the **HTML** pages.

Absolute XPaths are not advisable for most of the time due to following reasons

1. Absolute XPaths are lengthier and hence they are not readable
2. Absolute XPaths are not resilient. They tend to break when minor structural changes are introduced to the web pages

Absolute XPaths shall be used only when a relative XPath cannot be constructed. (highly unlikely). **It is not recommended to use absolute XPath in Selenium.**

**Syntax:** Absolute XPaths start with `/html`

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

### Relative XPath

Relative XPaths is used for locating elements with respect to a element with known (solid) XPath. The element of your choice is referred relative to a known element.

**Syntax:** Relative XPaths are started with two forward slashes `//`.

**Examples :**

1. `//div[@id='divUsername']/input`
2. `//form/div[@id='divUsername']/input`

### 3. **//form/\*/input**

There could be zero or more elements between the **context element** (starting element with a known Xpath) and the target element

#### NOTES:

1. *Absolute XPaths are faster than the relative XPaths*
2. *Use shortest possible relative XPaths*

## What should be considered when choosing an XPath?

It is important to consider following while choosing an XPath from available options.

A good locator is:

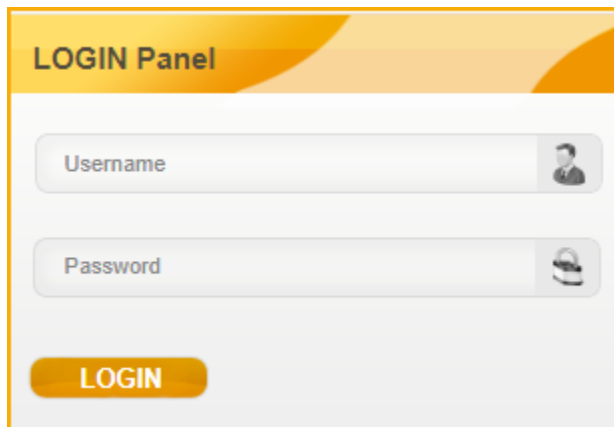
- Unique
- Descriptive
- Resilient
- Shorter in length

When you want to locate a single element your XPath should have only one candidate element (**unique**). It will be easier to identify the element if it is descriptive and short (for **maintainability**). XPaths generated from tools may not be **user-friendly**. It should be possible to locate an element with given XPath when the test is run again in subsequent releases too. XPath should be selected in such a way it is valid even after changes in DOM (**resilient**). You will have multiple XPath options. A shorter XPath shall be selected to make it more readable in your test scripts.

## Sample HTML code

Following HTML (**DOM**) will be used for the explaining most of the XPath syntax in this post.

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>...</head>
  <body>
    <div id="wrapper">
      <div id="content">
        <style type="text/css">...</style>
        <div id="divLogin">
          <div id="divLogo">...</div>
          <form id="frmLogin" method="post" action="/index.php/auth/validateCredentials">
            <div id="logInPanelHeading">LOGIN Panel</div>
            <div id="divUsername" class="textInputContainer">
              <input name="txtUsername" id="txtUsername" type="text">
              <span class="form-hint">Username</span>
            </div>
            <div id="divPassword" class="textInputContainer">
              <input name="txtPassword" id="txtPassword" type="password">
              <span class="form-hint">Password</span>
            </div>
            <div id="divLoginHelpLink"></div>
            <div id="divLoginButton">
              <input type="submit" name="Submit" class="button" id="btnLogin" value="LOGIN">
            </div>
          </form>
        </div>
      </div>
    </div>
  </body>
</html>
```



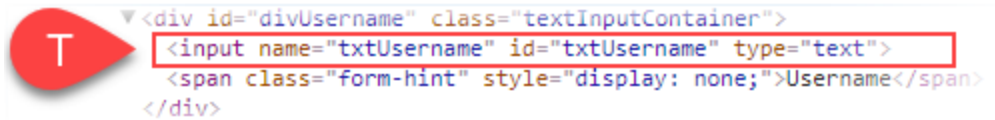
## Locating an Elements with a Known Attribute

The following syntax can be used for locating elements when at least one of the attribute's value is unique and static.

**Syntax:**

```
//*[@attributeName='value']
```

Let's locate the **username** field in following.



We have **three** valid XPath

#### Examples :

1. `//*[@id='txtUsername']`
2. `//*[@name='txtUsername']`
3. `//*[@type='text']`

**NOTE:** Third XPath **should not** be used even though it is a valid XPath. Because It will not be a unique XPath in most of the cases. There will be many elements with `type='text'`. Hence Selenium will not be able to locate the target element uniquely.

**Single quotations** should be used to enclose the values (in Java). You need to use escape character if you wish to enclose the values with double quotes.

**Example :** `//*[@id=\"txtUsername\"]`

In our real life we cannot locate a person with name **Mohammed** in a Muslim community.

## How does Selenium works when there are multiple elements (candidates) for an XPath?

Selenium will pick the first element in the path if there are multiple candidates for a given XPath when `webdriver.findElement(By.xpath("XPath"))` method is used. All the candidate elements can be assigned to a List when `webdriver.findElements(By.xpath("XPath"))` method is used.

#### Examples :

1. To select the third input element : `//form/input[3]`
2. To select the last input element : `//form/input[last()]`

Locating elements by position is discussed further in a separate section.

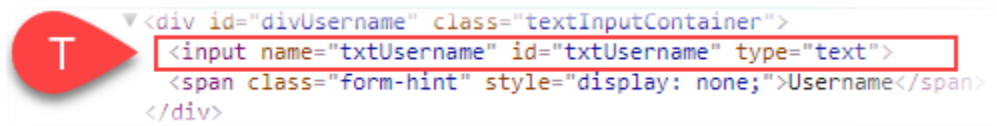
**findElement** method throws **NoSuchElementException** error when there is no elements with given XPath. Use **findElements** method and check the size when working with non-present elements.

## Locating Elements with a Tag-name and an Attribute

#### Syntax:

`//tagName[@attributeName='value']`

Let's consider locating the username input field again



### Examples :

1. `//input[@id='txtUsername']`
2. `//input[@name='txtUsername']`

This is one of the most commonly used Xpath. Most of the plugs can generate above XPath's automatically

## Locating Elements with static Visible Text (exact match)

Following syntax is used for locating elements containing exact text within opening tag and closing tag (**inner text**).

### Syntax:

`//tagName[text()='exact text']` or

`/*[text()='exact text']`

Let's consider locating following hyperlink



### Examples :

1. `//a[text()='Pragmatic']`
2. `/*[text()='Pragmatic']`

**NOTE:** *The inner text is case sensitive.*

Locating elements by the visible text is not advisable

1. If you are testing a multilingual application
2. When same text is appearing in more than one location.

## Locating Elements when part of the visible text is static (partial match)

### Syntax:

```
//tagName[contains(text(),'substring')]
```

```
//tagName[contains(.,'substring')]
```

```
//*[contains(text(),'substring')]
```



### Examples :

1. `//a[contains(text(),'Pragmatic')]`
2. `//a[contains(., 'Test Labs')]`
3. `//*[contains(text(), 'Test Labs')]`

Validate the XPath syntax before running the test scripts. Validating the XPath is discussed in a separate section.

## Locating Elements when prefix of the inner text is static

You can locate the elements when part of the starting text of the inner text are static.

### Syntax :

```
//tagName[starts-with(text(),'Prefix of Inner Text')]
```

```
//*[starts-with(text(),'Prefix of Inner Text')]
```



### Examples :

1. `//a[starts-with(text(),'Pragmatic')]`
2. `//*[starts-with(text(), 'Prag')]`

## Locating elements with Visible text in input elements

Locating input elements with visible text should not be confused with the locating elements with inner text as in above sections. Attribute value should be used for locating the visible text in input type elements.

### Syntax :



1. //tagName[@value='visibleText']

```
<!DOCTYPE html>
<html>
<body>
```

```
<form action="/action_page.php">
First <input type="text" name="FirstName" value="Janesh">
<br>
Last name: <input type="text" name="LastName" value="Kodikara">
```

First name:

Last name:

Click the "Submit" button and the for

## Examples :

1. //input[@value='Janesh']

We have already discuss the process of locating elements by tag-name and an attribute.

## Locating Elements with Multiple Attributes

Sometimes it may not be possible to locate an element with a single attribute uniquely as there could be more than one candidate elements with given attribute. In the real world, we have a similar scenarios. We cannot locate a person by just their first name or last name alone. We will have to use a combination of first name and last name to locate a person uniquely without making any confusion.

Similar technique is used in Selenium for locating elements when there are more than one elements with a given attribute. We will use two or more attributes together to locate an element uniquely.

### Syntax :

//\*[@attribute1='value1'][attribute2='value2']...[attributeN='valueN'] or  
//tagName[attribute1='value1'][attribute2='value2']...[attributeN='valueN'] or  
//\*[@attribute1='value1' and attribute2='value2']  
//tagName[attribute1='value1' and attribute2='value2']

```
<div id="divLoginButton">
  <input type="submit" name="Submit" class="button" id="btnLogin"
  value="LOGIN">
</div>
```

## Examples :

1. //\*[@type='submit'][@value='LOGIN']
2. //input[@class='button'][@type='submit'][@value='LOGIN'][@name='Submit']
3. //\*[@type='submit' and @value='LOGIN']

## Locating Elements with Dynamic Attribute values

Following syntax could be used when a part of the attribute's values are NOT changed. We can use the non changing value for locating the element.

### Syntax :

`//elementName[contains(@attributeName,'substring of the value')]` or

`//*[contains(@attributeName,'substring of the value')]`

`//elementName[starts-with(@attributeName,'fixed prefix of the value')]`



### Examples :

1. `//a[contains(@href,'pragmatic')]`
2. `//*[contains(@href,'testlabs')]`
3. `//a[starts-with(@href,'pragmatic')]`

The `ends-with()` function is part of XPath 2.0. Most of the browsers do not support Xpath 2.0 at the time of the writing.

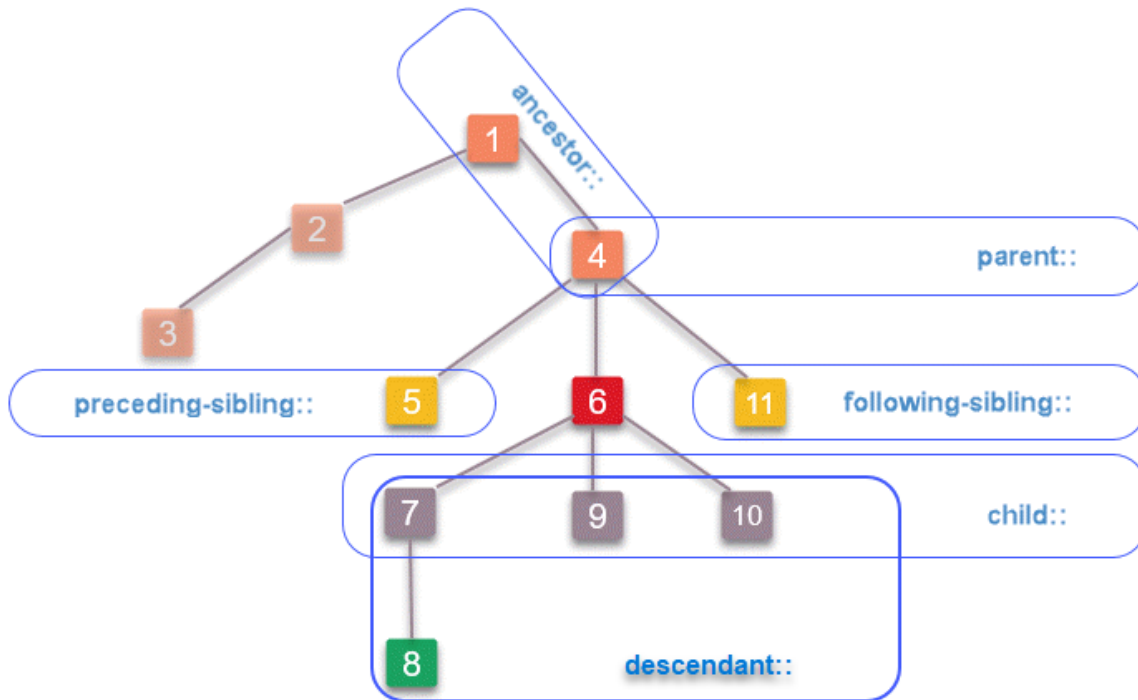
## Locating elements relative to known element

If our **target elements** do not have unique attribute(s) or static innerHTML, we need to locate the elements with respect to an element who has an **unchanging XPath**.

We do this very well in our real life. We always give direction to an unknown location with respect to a well known location (a **landmark**). Giving direction to your home from a well-known shop, statue, railway station etc.

XPath has thirteen (**13**) different axes. We will look into a subset of useful axes in this blog post which can be used with Selenium.

Your target element should be closer to the known element (**context element**) to make XPath shorter, resilient and readable.



## Locating a parent element

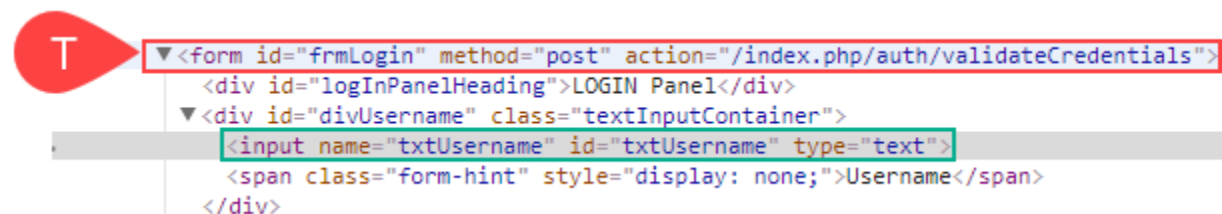
The parent axis contains the parent of the context node. Every context element has only one parent element except root element (html).

### Syntax :

```
//<knownXPath>/parent::* or
//<knownXPath>/parent::elementName
//<knownXPath>/..
```

Let's see how to locate the `form` element with respect to the username field. We need to select an element with unchanging XPath. In this case we will take the username field.

XPath of the known element : `//input[@id='txtUsername']`



### Examples :

1. `//input[@id='txtUsername']/parent::form`
2. `//input[@id='txtUsername']/parent::*`
3. `//input[@id='txtUsername']/..`

There can be only one parent to a context (known) element. Hence specifying the element name is optional. But it is good to specify the element name for readability.

## Locating a child element

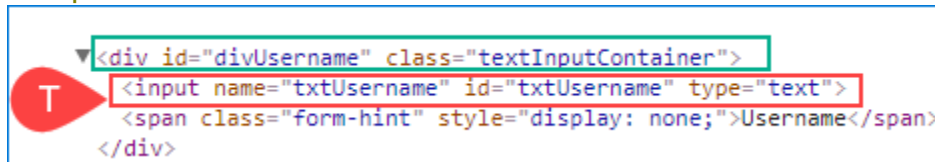
The child axis contains the children of the context node

**Syntax :**

`//<xpathOfContextElement>/child::<elementName>` or

`//<xpathOfContextElement>/child::*`

`//<xpathOfContextElement>/<elementName>`



**Examples :**

In following examples context element's XPath is `div[@id='divUsername']`

1. `//div[@id='divUsername']/child::input`
2. `//div[@id='divUsername']/input`

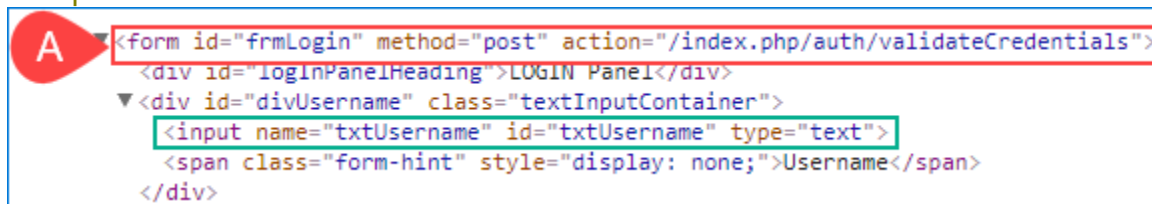
In practice `/` is used instead of `child::` from the known XPath.

## Locating grand children

**Syntax :**

`//<xpathOfContextElement>/*/<elementName>`

`//<xpathOfContextElement>/child/<elementName>`



**Examples :**

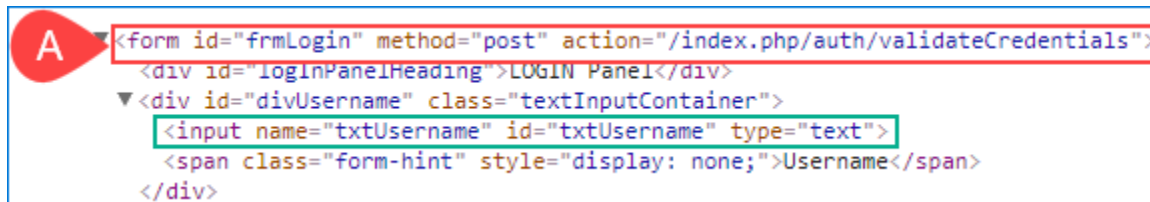
1. `//form/*/input`
2. `//form/div/input`

## Locating ancestors of a known element

The ancestor axis contains the ancestors of the known element; ancestor axis consists of the parent of a known element and the parent's parent so on.

#### Syntax :

`//<xpathOfContextElement>/ancestor::<elementName>` or `//<xpathOfContextElement>/ancestor::*`



#### Examples :

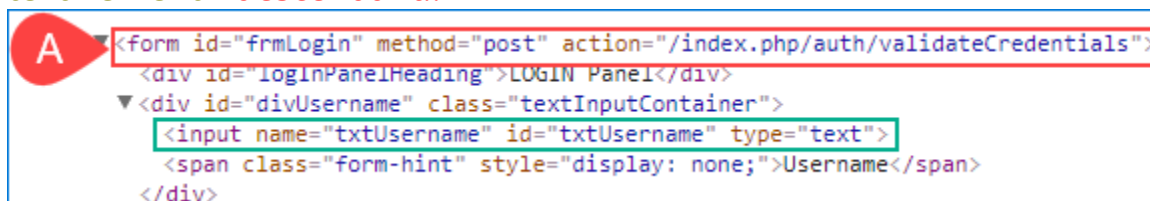
1. `//input[@id='txtUsername']/ancestor::form` : will select the `form` element
2. `//input[@id='txtUsername']/ancestor::*` : `div` element will be selected from the available candidates (`div`, `form` etc) as it comes first in the path if you use `findElement` method.

## Locating descendants of a known element

The descendant axis contains the descendants of a known element; descendant axis consists of the children of a context element and their children and so on.

#### Syntax :

`//<xpathOfContextElement>/descendant::<elementName>` or `//<xpathOfContextElement>/descendant::*`



#### Examples :

1. `//form[@id='frmLogin']/descendant::input`
2. `//form[@id='frmLogin']//input`

You can use `//` instead of `descendant::` keyword to locate descendants.

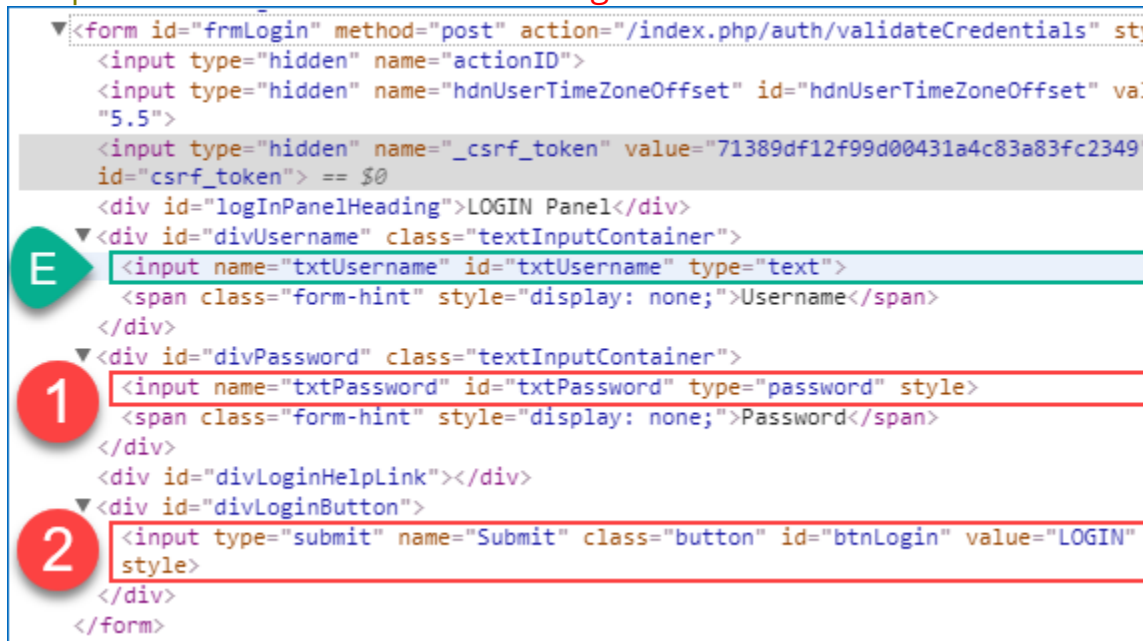
## Locating following elements

Keyword **following::** can be used for locating element(s) anywhere below the tree with respect to a known element (context element).

**Syntax :**

`//<xpathOfContextElement>/following::<elementName>` or

`//<xpathOfContextElement>/following::*`



**Examples :**

1. `//input[@id='txtUsername']/following::input`
2. `//input[@id='txtUsername']/following::*`

There are two candidate elements. Any descendant elements after the first candidate in the path are excluded by Selenium when you use `findElement` method.

To select the login button input element with respect to the username field.

1. `//input[@id='txtUsername']/following::input[last()]`
2. `//input[@id='txtUsername']/following::input[2]`

## Locating preceding element

Keyword **preceding::** is used for locating an element before a known (XPath) element.

The **preceding** axis contains all nodes that are descendants of the root of the tree in which the context node is found, are not ancestors of the context node, and occur before the context node in document order

**Syntax :**

`//<xpathOfContextElement>/preceding::<elementName>` or  
`//<xpathOfContextElement>/preceding::*`

```
▼ <div id="divUsername" class="textInputContainer" style="display: none;">
  <input name="txtUsername" id="txtUsername" type="text" style="display: none;">
  <span class="form-hint" style="display: none;">Username</span>
</div>
▼ <div id="divPassword" class="textInputContainer" style="display: none;">
  <input name="txtPassword" id="txtPassword" type="password" style="display: none;">
  <span class="form-hint" style="display: none;">Password</span>
</div>
<div id="divLoginHelpLink"></div>
```

### Examples :

1. `//span[text()='Password']/preceding::input`

There will be two candidate elements (username and password elements). Selenium will select the password input element when findElement method is used. Elements are ordered from the context element (**span**).

2. `//span[text()='Password']/preceding::input[2]`

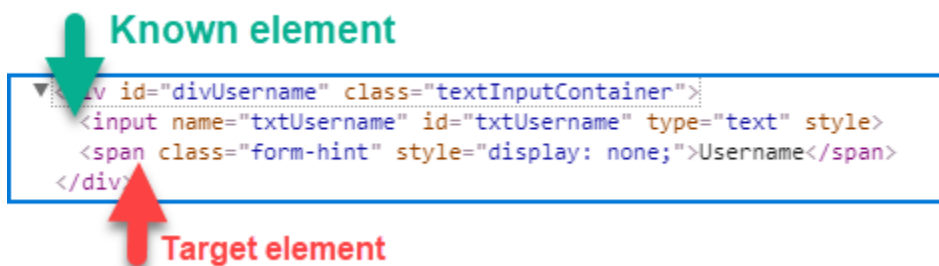
Above can be used for selecting the username field.

## Locating following sibling

Keyword **following-sibling::** is used to locate the element(s) comes after a context element within same HTML hierarchy. Following siblings are the elements (children) of the context node's parent that occur after the context element in document order

### Syntax :

`//<xpathOfContextElement>/following-sibling::<elementName>` or  
`//<xpathOfContextElement>/following-sibling::*`



### Examples :

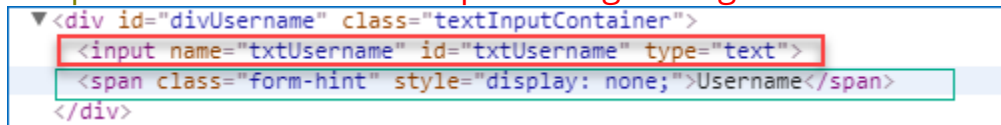
1. `//*[@id='txtUsername']/following-sibling::span`
2. `//*[@id='txtUsername']/following-sibling::*`

## Locating preceding sibling

Keyword **preceding-sibling::** is used to select the sibling(s) that comes before the context node with a known XPath, those elements (children) of the context node's parent that occur before the context element in document order.

**Syntax :**

`//<xpathOfKnownElement>/preceding-sibling::<elementName>` or  
`//<xpathOfKnownElement>/preceding-sibling::*`



**Examples :**

1. `//span[contains(text(),'Username')]/preceding-sibling::input`
2. `//span[contains(text(),'Username')]/preceding-sibling::*`

With this we complete discussion of XPath with axes. Please note that we have not discussed attribute, ancestor-or-self, descendant-or-self, namespace and self **axes** in this article as they do not have practical usage in the context of Selenium.

## Locating Several XPaths

You can locate multiple elements by separating two or more XPath expressions with | character.

**Syntax :**

`XPath1 | Xpath2.... | XPathN`

If the first XPath is available in the first element is selected by Selenium for further actions. If both are available first one is Selected when you use `findElement` method. Both will be selected if you use `findElements` method. If only second XPath (Xpath2) is available then second element will be selected. If both of them are NOT available Selenium gives an error, `NoSuchElementException` when you use `findElement` method.

**Example :**

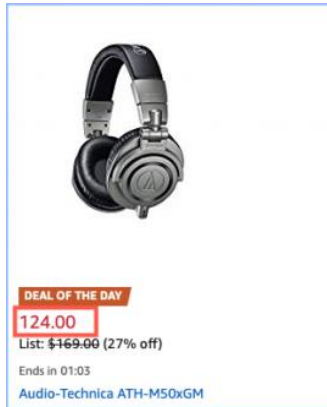
1. `//input[@id='txtUsername']|//input[@name='txtPassword']`  
In this example available elements can be located.  
This is useful when you know one of them would exist when the page is loaded.
2. `//*[@id='txtUsername']|//*[@name='txtPassword']|*[@name='btnLogin']`

## Working with Operators



You can use various operators to compare numeric value in attributes and inner text. We have addition ( + ), subtraction ( - ), multiplication ( \* ), division ( div ), equal ( = ), not equal ( != ), less than ( < ), less than or equal ( <= ), greater than ( > ), greater than or equal ( >= ), **and**, or , mod operators.

Lets see an example



... `<span class="a-size-medium inlineBlock unitLineHeight dealPriceText">124.00</span> == $0`

`::after`

`</div>`

▶ `<div class="a-row a-spacing-top-mini unitLineHeight">...</div>`

`::after`

`</div>`

▶ `<div class="a-row a-spacing-mini">...</div>`

▶ `<div class="a-row a-spacing-mini">...</div>`

▶ `<div class="a-row a-spacing-mini">...</div>`

▶ `<div class="a-row a-spacing-mini">...</div>`

... div div div div div span.a-size-medium.inlineBlock.unitLineHeight.dealPriceText

`//span[text()>100]` 1 of 30 **Cancel**

Say you need to locate element(s) with deal price greater than 100

`//span[contains(@class,'dealPriceText') and text()>100]`

You can combine the techniques learned to build complex XPaths and locate any element in DOM.