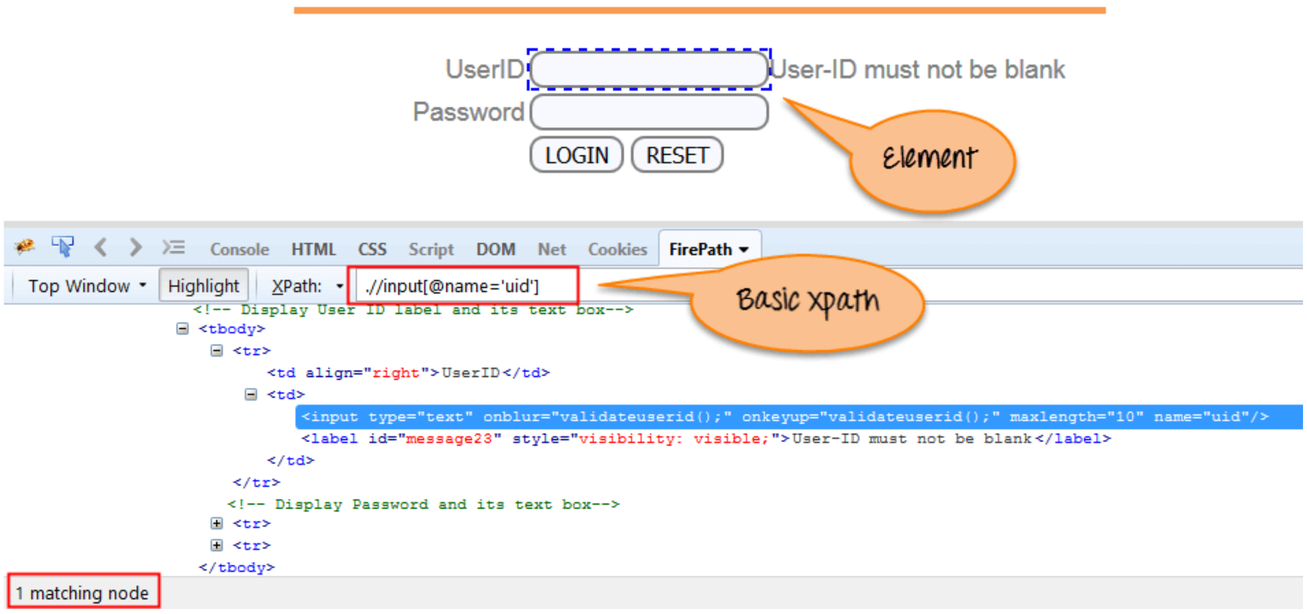
How To Write Dynamic XPath In Selenium WebDriver

1) Basic XPath:

XPath expression select nodes or list of nodes on the basis of attributes like **ID , Name, Classname**, etc. from the XML document as illustrated below.

Xpath=//input[@name='uid']

Here is a link to access the page http://demo.guru99.com/test/selenium-xpath.html



```
.//input[@name='uid']
```

```
<!-- Display User ID label and its text box-->
⊟ <tbody>
  ⊟ <tr>
      <td align="right">UserID</td>
    ⊟ <td>
        <input type="text" onblur="validateuserid();" onkeyup="validateuserid();" maxlength="10" name="uid"/>
        <label id="message23" style="visibility: visible;">User-ID must not be blank</label>
      </td>
    </tr>
   <!-- Display Password and its text box-->
  ⊞ <tr>
  ⊞ <tr>
  </tbody>
```

1 matching node

Basic XPath

Some more basic xpath expressions:

Xpath=//input[@type='text']
Xpath=//label[@id='message23']
Xpath=//input[@value='RESET']
Xpath=//*[@class='barone']
Xpath=//a[@href='http://demo.guru99.com/']
Xpath=//img[@src='//guru99.com/images/home/java.png']

2) Contains():

Contains() is a method used in XPath expression. It is used when the value of any attribute changes dynamically, for example, login information.

The contain feature has an ability to find the element with partial text as shown in below XPath example.

In this example, we tried to identify the element by just using partial text value of the attribute. In the below XPath expression partial value 'sub' is used in place of submit button. It can be observed that the element is found successfully.

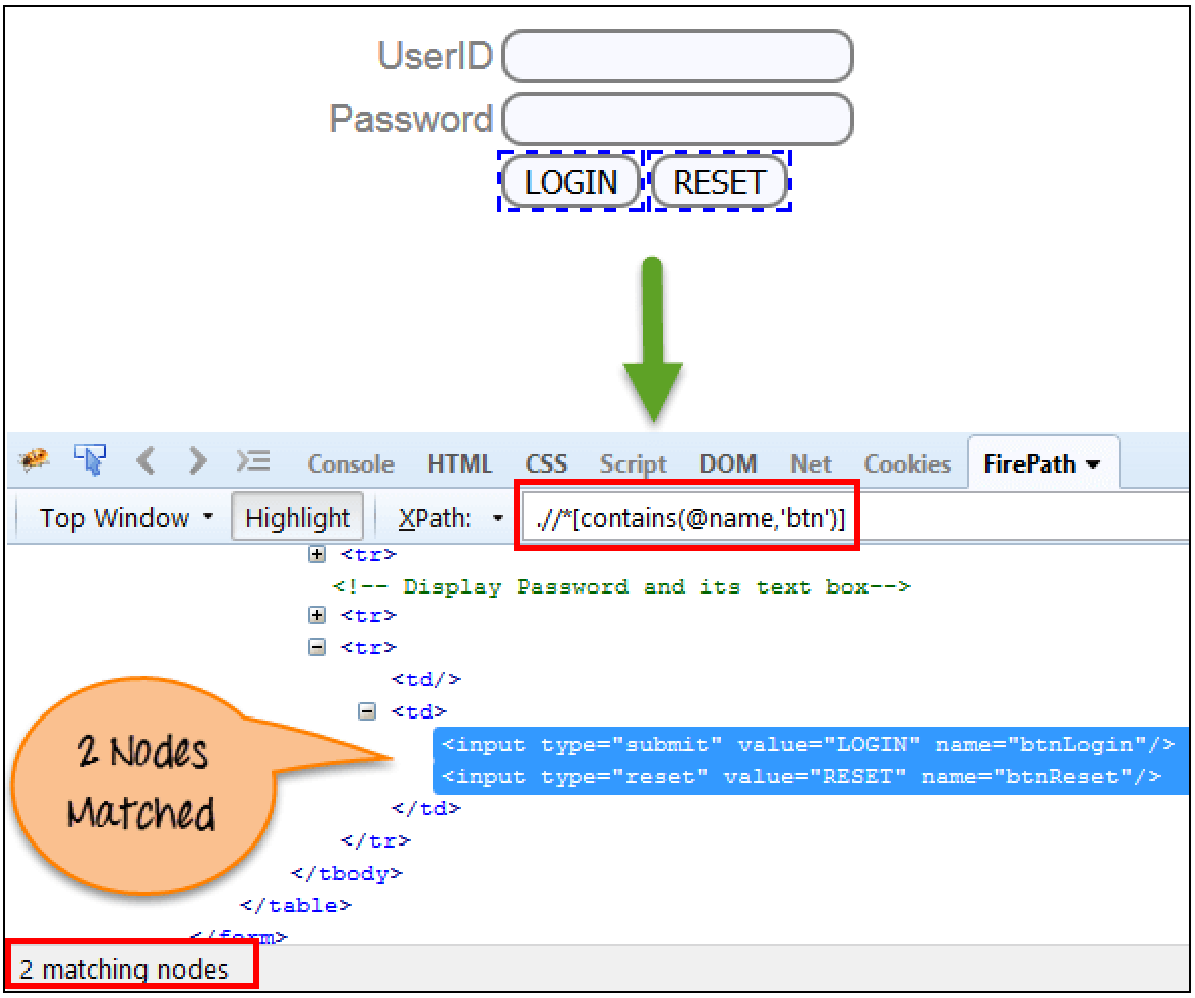
Complete value of 'Type' is 'submit' but using only partial value 'sub'.

```
Xpath=//*[contains(@type,'sub')]
```

Complete value of 'name' is 'btnLogin' but using only partial value 'btn'.

```
Xpath=//*[contains(@name,'btn')]
```

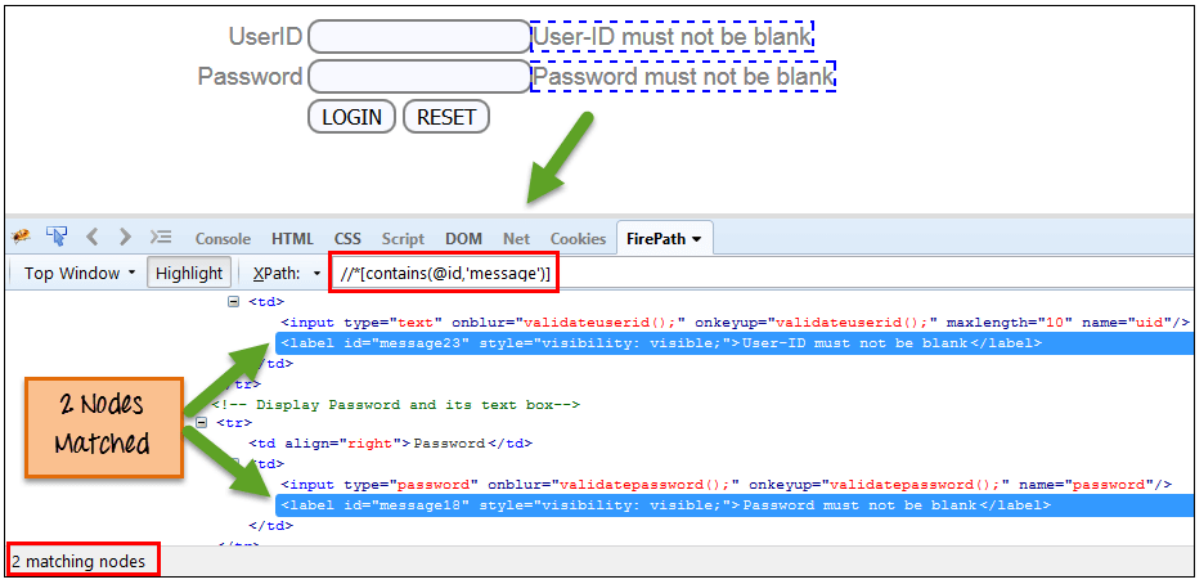
In the above expression, we have taken the 'name' as an attribute and 'btn' as an partial value as shown in the below screenshot. This will find 2 elements (LOGIN & RESET) as their 'name' attribute begins with 'btn'.



```
.//*[contains(@name,'btn')]
<tr>
  <!-- Display Password and its text box-->
<tr>
<tr>
    <td/>
    <td>
      <input type="submit" value="LOGIN" name="btnLogin"/>
      <input type="reset" value="RESET" name="btnReset"/>
    </td>
  </tr>
</tbody>
</table>
</form>
2 matching nodes
```

Similarly, in the below expression, we have taken the 'id' as an attribute and 'message' as a partial value. This will find 2 elements ('User-ID must not be blank' & 'Password must not be blank') as its 'id' attribute begins with 'message'.

```
Xpath=//*[contains(@id,'message')]
```



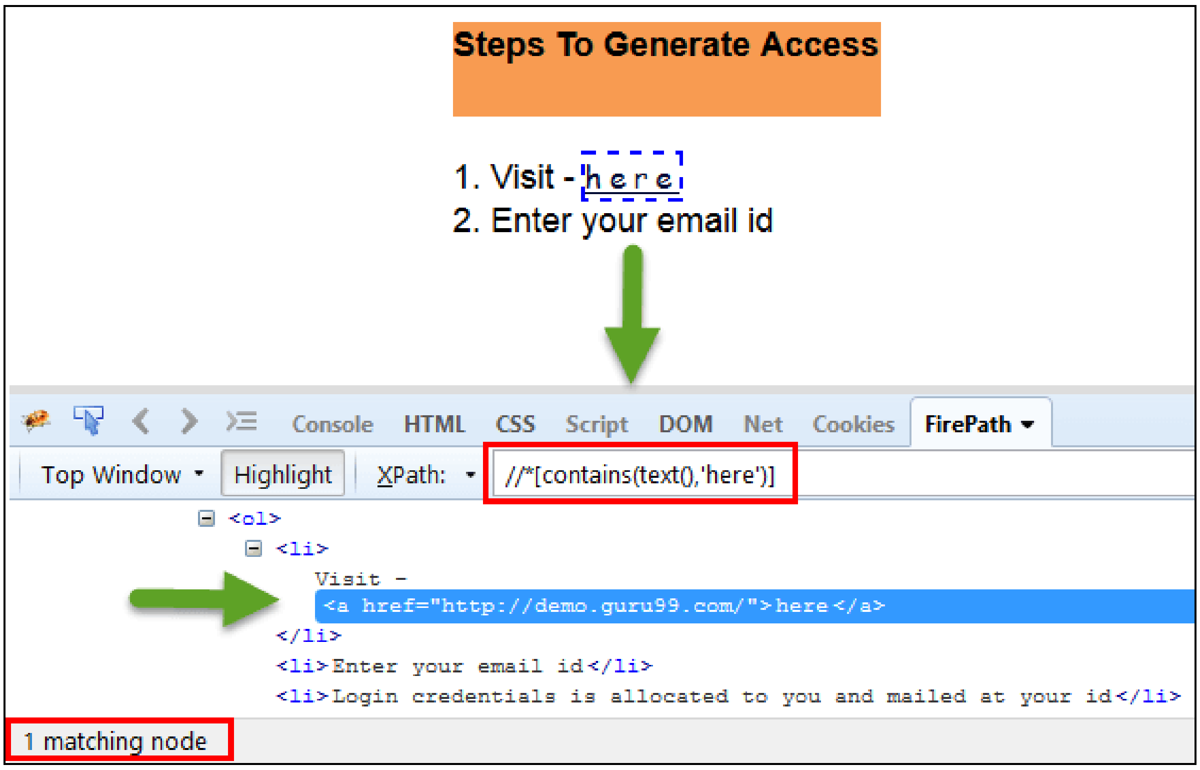
```
//*[contains(@id,'message')]
```

```
<td>
    <input type="text" onblur="validateuserid();" onkeyup="validateuserid();" maxlength="10" name="uid"/>
    <label id="message23" style="visibility: visible;">User-ID must not be blank</label>
</td>
</tr>
<!-- Display Password and its text box-->
<tr>
  <td align="right">Password</td>
  <td>
    <input type="password" onblur="validatepassword();" onkeyup="validatepassword();" name="password"/>
    <label id="message18" style="visibility: visible;">Password must not be blank</label>
  </td>
```

2 matching nodes

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')]



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

```
<ol>
  <li>
    Visit -
    <a href="http://demo.guru99.com/">here</a>
  </li>
  <li>Enter your email id</li>
  <li>Login credentials is allocated to you and mailed at your id</li>
```

1 matching node

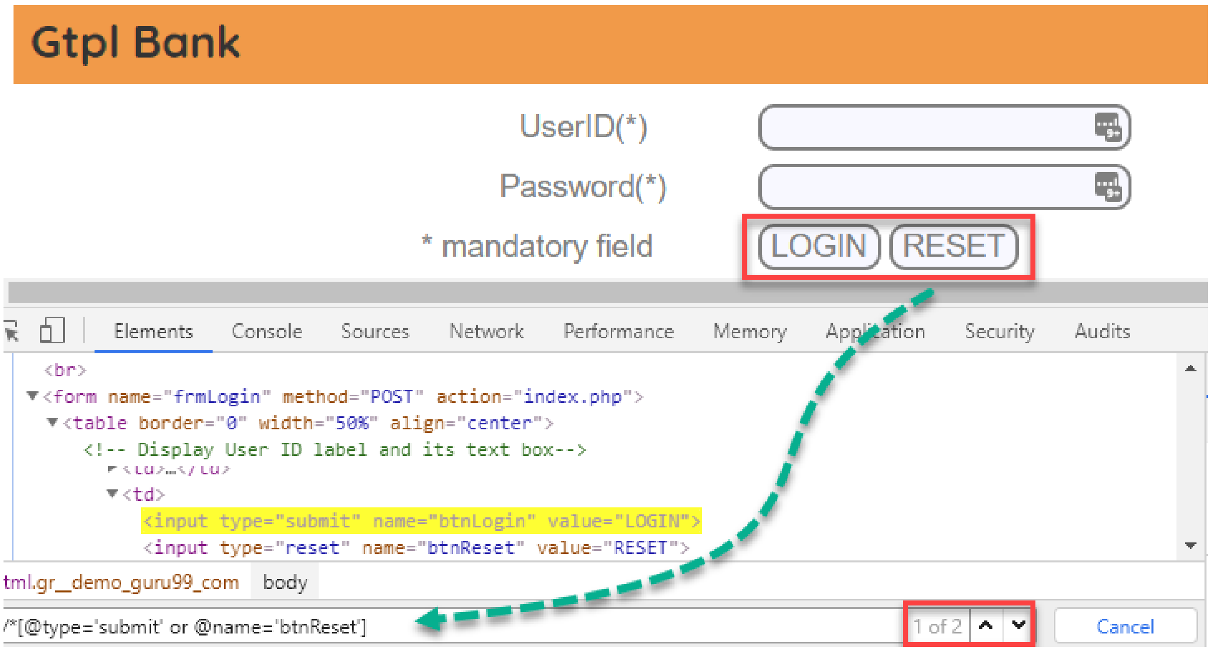
3) Using OR & AND:

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.

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

Highlighting both elements as “LOGIN ” element having attribute ‘type’ and “RESET” element having attribute ‘name’.



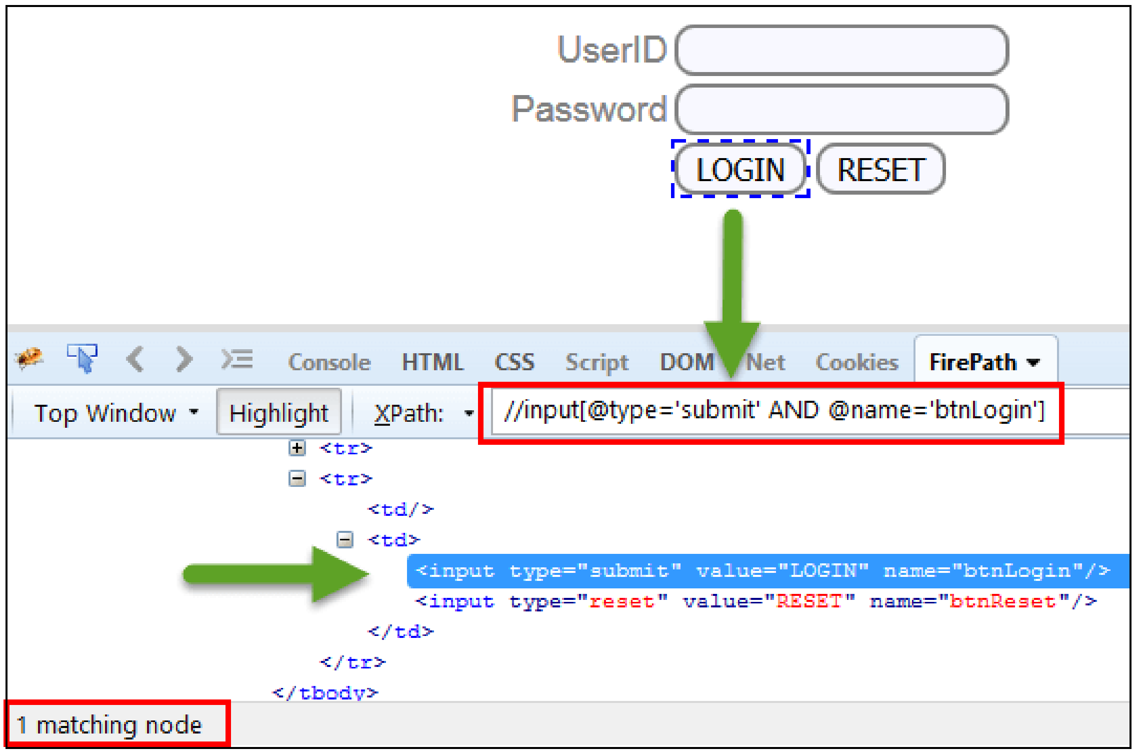
```
<br>
▼<form name="frmLogin" method="POST" action="index.php">
  ▼<table border="0" width="50%" align="center">
      <!-- Display User ID label and its text box-->
      ►<tr>…</tr>
      ▼<td>
          <input type="submit" name="btnLogin" value="LOGIN">
          <input type="reset" name="btnReset" value="RESET">
```

```
/*[@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.

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

In below expression, highlighting ‘LOGIN’ element as it having both attribute ‘type’ and ‘name’.



```
//input[@type='submit' AND @name='btnLogin']
```

```
<tr>
<tr>
    <td/>
    <td>
        <input type="submit" value="LOGIN" name="btnLogin"/>
        <input type="reset" value="RESET" name="btnReset"/>
    </td>
</tr>
</tbody>
```

1 matching node

4) Xpath Starts-with

XPath starts-with() is a function used for finding the web element whose attribute value gets changed on refresh or by other dynamic operations on the webpage. In this method, the starting text of the attribute is matched to find the element whose attribute value changes dynamically. You can also find elements whose attribute value is static (not changes).
For example -: Suppose the ID of particular element changes dynamically like:

Id=" message12″

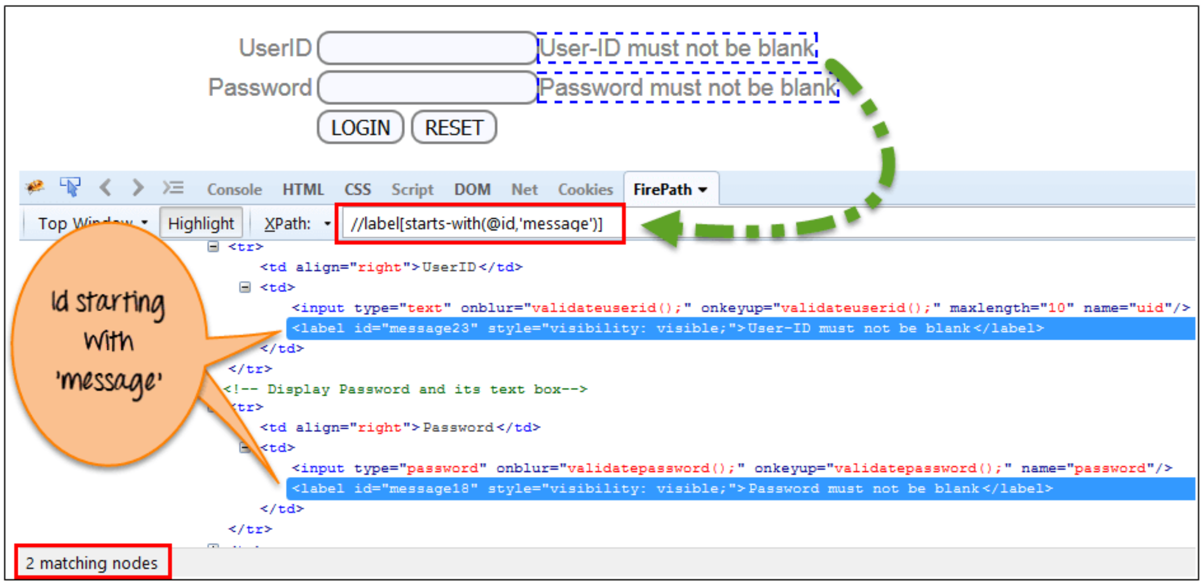
Id=" message345″

Id=" message8769″

and so on.. but the initial text is same. In this case, we use Start-with expression.

In the below expression, there are two elements with an id starting "message"(i.e., 'User-ID must not be blank' & 'Password must not be blank'). In below example, XPath finds those element whose 'ID' starting with 'message'.

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


UserID
Password
User-ID must not be blank
Password must not be blank
LOGIN
RESET
Console HTML CSS Script DOM Net Cookies FirePath
Top Window
Highlight
XPath:
Id starting with 'message'

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

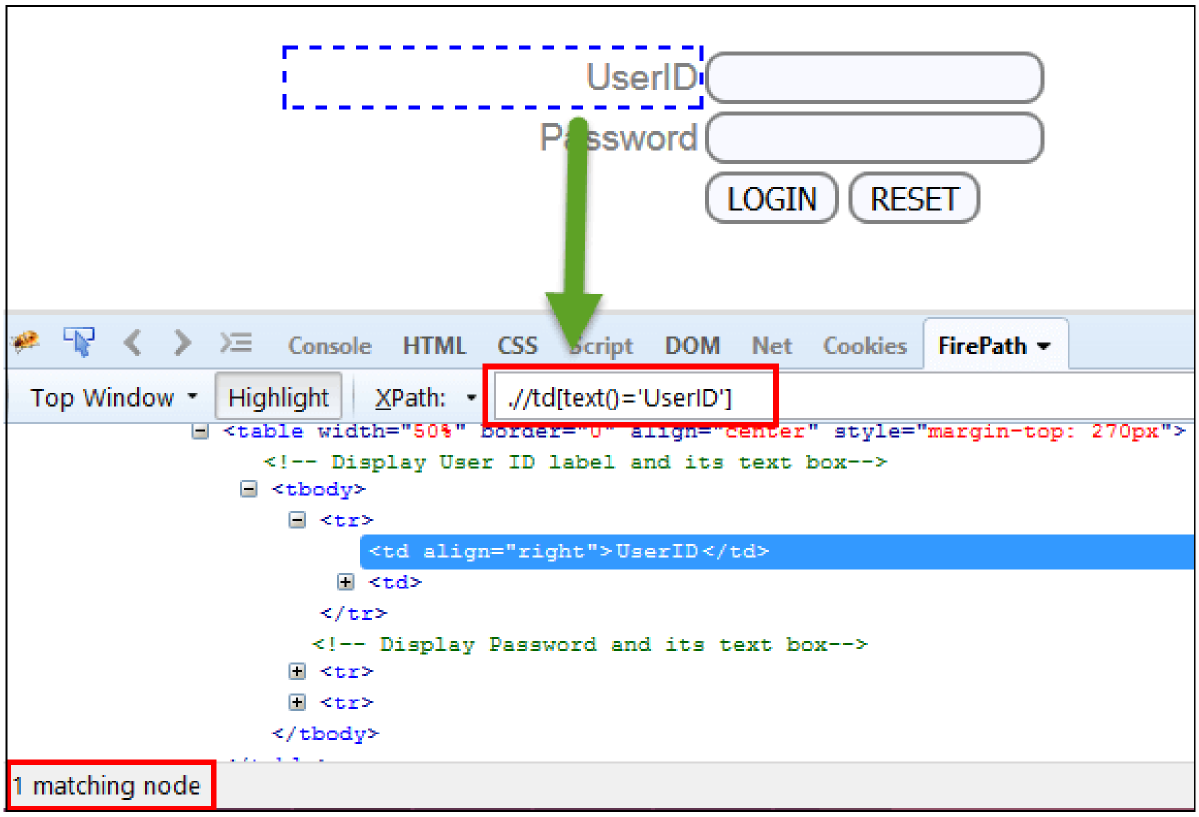
```
<tr>
    <td align="right">UserID</td>
    <td>
        <input type="text" onblur="validateuserid();" onkeyup="validateuserid();" maxlength="10" name="uid"/>
        <label id="message23" style="visibility: visible;">User-ID must not be blank</label>
    </td>
</tr>
<!-- Display Password and its text box-->
<tr>
    <td align="right">Password</td>
    <td>
        <input type="password" onblur="validatepassword();" onkeyup="validatepassword();" name="password"/>
        <label id="message18" style="visibility: visible;">Password must not be blank</label>
    </td>
</tr>
```

2 matching nodes

5) XPath Text() Function

The **XPath text() function** is a built-in function of selenium webdriver which is used to locate elements based on text of a web element. It helps to find the exact text elements and it locates the elements within the set of text nodes. The elements to be located should be in string form.
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 “UserID”.

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



```
.//td[text()='UserID']
<table width="50%" border="0" align="center" style="margin-top: 270px">
  <!-- Display User ID label and its text box-->
  <tbody>
    <tr>
      <td align="right">UserID</td>
      <td>
    </tr>
    <!-- Display Password and its text box-->
    <tr>
    <tr>
  </tbody>
1 matching node
```

XPath axes methods:

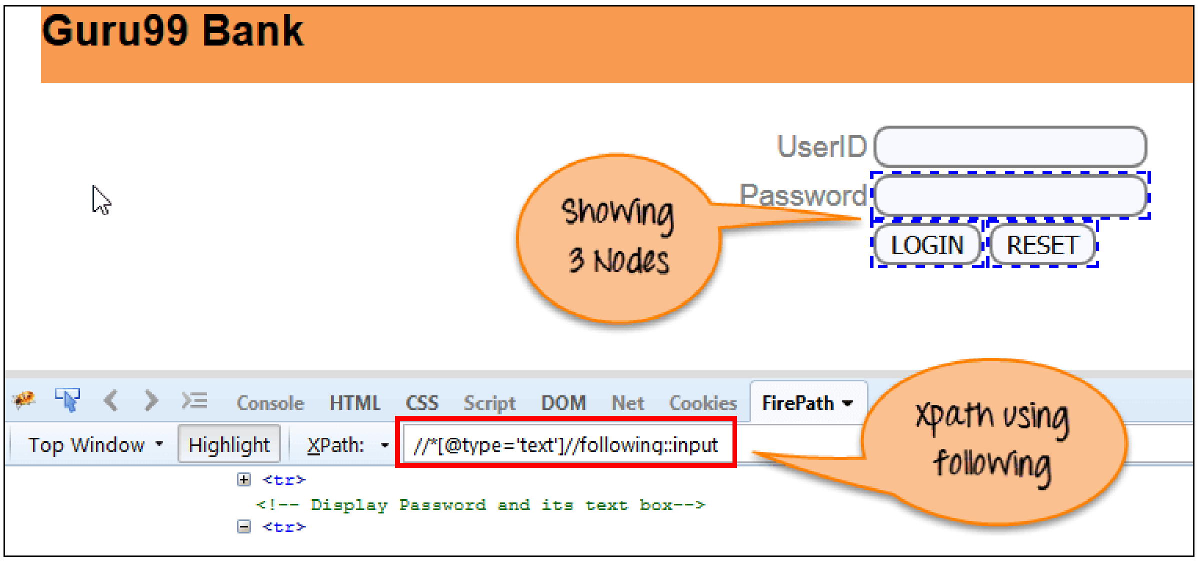
These XPath axes methods are used to find the complex or dynamic elements. Below we will see some of these methods.

For illustrating these XPath axes method, we will use the Guru99 bank demo site.

1) Following:

Selects all elements in the document of the current node( ) [UserID input box is the current node] as shown in the below screen.

`Xpath=//*[@type='text']//following::input`


Guru99 Bank
UserID
Password
LOGIN
RESET
Showing 3 Nodes
Console HTML CSS Script DOM Net Cookies FirePath
Top Window Highlight XPath:
Xpath using following

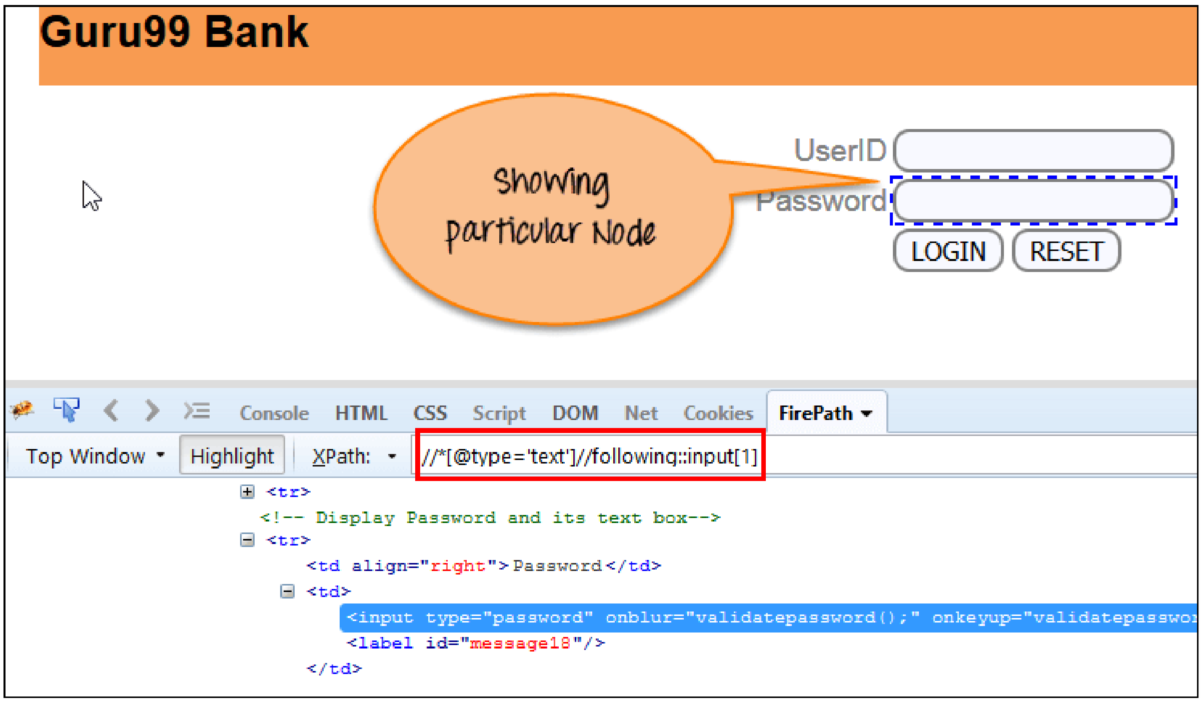
```
//*[@type='text']//following::input
⊞ <tr>
  <!-- Display Password and its text box-->
⊟ <tr>
```

There are 3 “input” nodes matching by using “following” axis- password, login and reset button. If you want to focus on any particular element then you can use the below XPath method:

```
Xpath=//*[@type='text']//following::input[1]
```

You can change the XPath according to the requirement by putting [1],[2]…………and so on.

With the input as ‘1’, the below screen shot finds the particular node that is ‘Password’ input box element.


Guru99 Bank
UserID
Password
LOGIN
RESET
Showing particular Node
Console HTML CSS Script DOM Net Cookies FirePath
Top Window Highlight XPath:

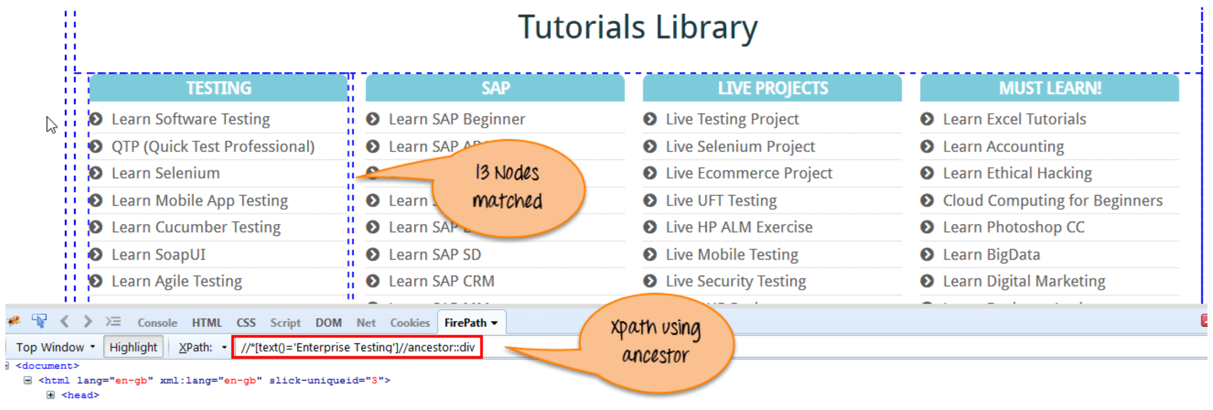
```
//*[@type='text']//following::input[1]
⊞ <tr>
  <!-- Display Password and its text box-->
⊟ <tr>
    <td align="right">Password</td>
  ⊟ <td>
      <input type="password" onblur="validatepassword();" onkeyup="validatepasswo
      <label id="message18"/>
    </td>
```

2) Ancestor:

The ancestor axis selects all ancestors element (grandparent, parent, etc.) of the current node as shown in the below screen.

In the below expression, we are finding ancestors element of the current node(“ENTERPRISE TESTING” node).

Xpath=//*[text()='Enterprise Testing']//ancestor::div



TESTING	SAP	LIVE PROJECTS	MUST LEARN!
Learn Software Testing	Learn SAP Beginner	Live Testing Project	Learn Excel Tutorials
QTP (Quick Test Professional)	Learn SAP A	Live Selenium Project	Learn Accounting
Learn Selenium		Live Ecommerce Project	Learn Ethical Hacking
Learn Mobile App Testing	Learn	Live UFT Testing	Cloud Computing for Beginners
Learn Cucumber Testing	Learn SAP	Live HP ALM Exercise	Learn Photoshop CC
Learn SoapUI	Learn SAP SD	Live Mobile Testing	Learn BigData
Learn Agile Testing	Learn SAP CRM	Live Security Testing	Learn Digital Marketing

```
//*[text()='Enterprise Testing']//ancestor::div
```

```
<document>
  <html lang="en-gb" xml:lang="en-gb" slick-uniqueid="3">
    <head>
```

There are 13 “div” nodes matching by using “ancestor” axis. If you want to focus on any particular element then you can use the below XPath, where you change the number 1, 2 as per your requirement:

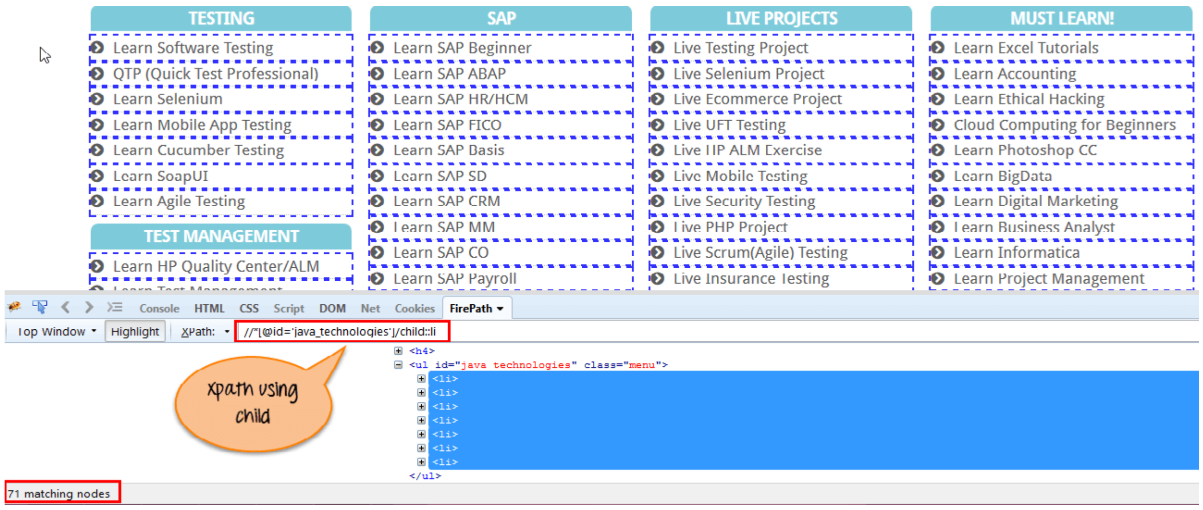
Xpath=//*[text()='Enterprise Testing']//ancestor::div[1]

You can change the XPath according to the requirement by putting [1], [2]…………and so on.

3) Child:

Selects all children elements of the current node (Java) as shown in the below screen.

Xpath=//*[@id='java_technologies']//child::li



//*[@id='java_technologies']/child::li

```
<h4>
<ul id="java technologies" class="menu">
    <li>
    <li>
    <li>
    <li>
    <li>
    <li>
    <li>
</ul>
```

71 matching nodes

There are 71 “li” nodes matching by using “child” axis. If you want to focus on any particular element then you can use the below xpath:

Xpath=//*[@id='java_technologies']//child::li[1]

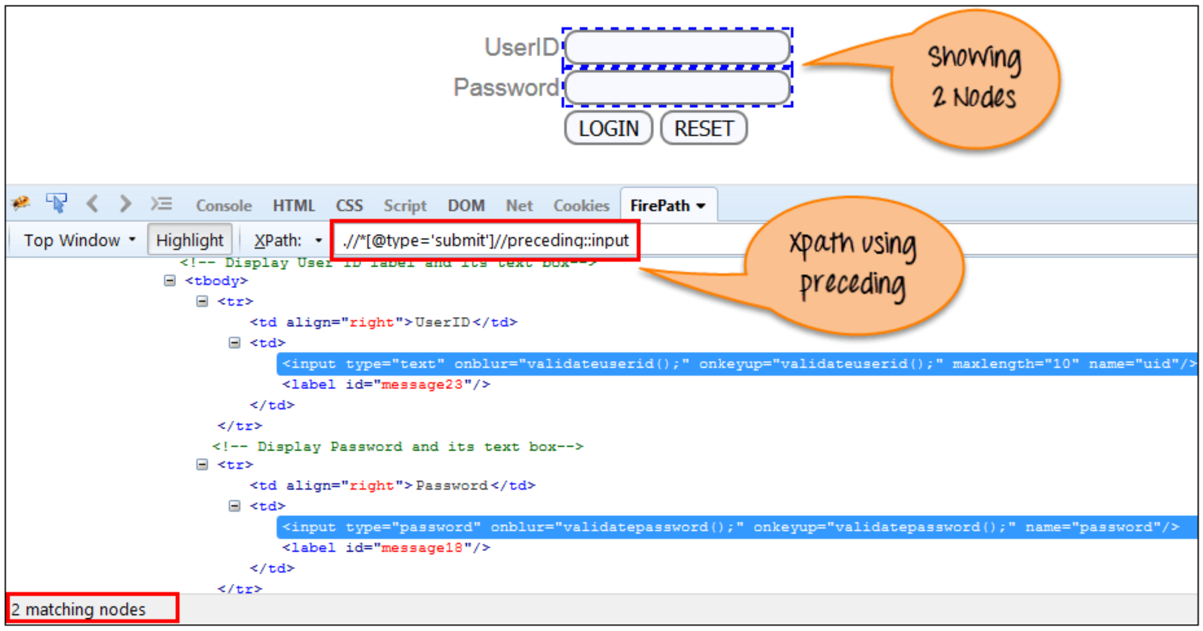
You can change the xpath according to the requirement by putting [1],[2]…………and so on.

4) Preceding:

Select all nodes that come before the current node as shown in the below screen.

In the below expression, it identifies all the input elements before “LOGIN” button that is **Userid** and **password** input element.

Xpath=//*[@type='submit']//preceding::input


UserID
Password
LOGIN
RESET
Showing 2 Nodes
Console HTML CSS Script DOM Net Cookies FirePath
Top Window Highlight XPath:
Xpath using preceding

.//*[@type='submit']//preceding::input

```
<!-- Display User ID label and its text box-->
<tbody>
    <tr>
        <td align="right">UserID</td>
        <td>
            <input type="text" onblur="validateuserid();" onkeyup="validateuserid();" maxlength="10" name="uid"/>
            <label id="message23"/>
        </td>
    </tr>
    <!-- Display Password and its text box-->
    <tr>
        <td align="right">Password</td>
        <td>
            <input type="password" onblur="validatepassword();" onkeyup="validatepassword();" name="password"/>
            <label id="message18"/>
        </td>
    </tr>
```

2 matching nodes

There are 2 "input" nodes matching by using "preceding" axis. If you want to focus on any particular element then you can use the below XPath:

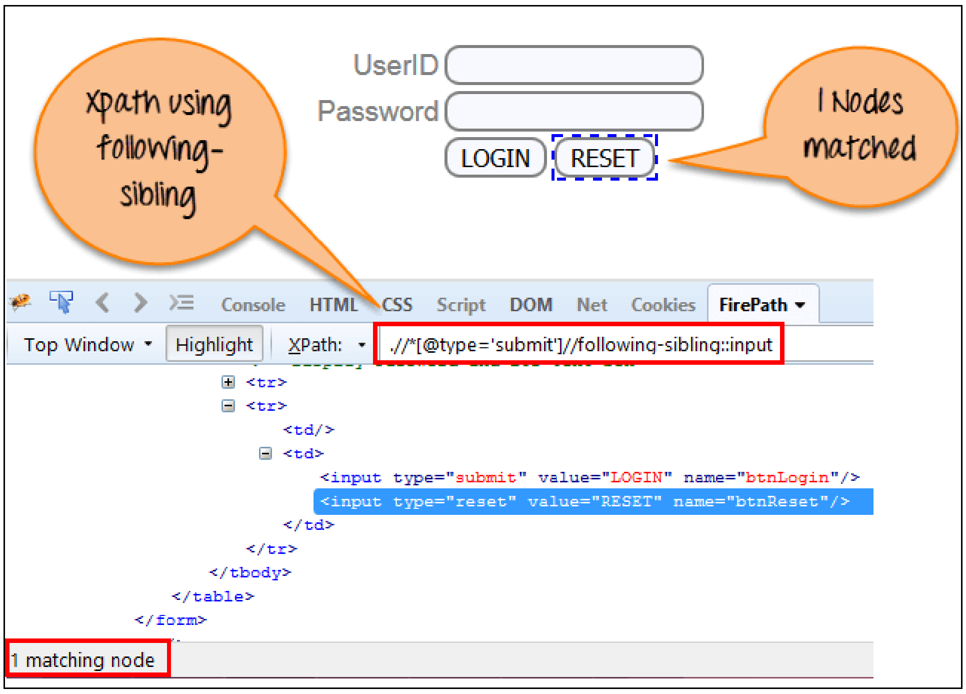
```
Xpath=//*[@type='submit']//preceding::input[1]
```

You can change the xpath according to the requirement by putting [1],[2]…………and so on.

5) Following-sibling:

Select the following siblings of the context node. Siblings are at the same level of the current node as shown in the below screen. It will find the element after the current node.

```
xpath=//*[@type='submit']//following-sibling::input
```


Xpath using following-sibling
1 Nodes matched
UserID
Password
LOGIN
RESET
Console HTML CSS Script DOM Net Cookies FirePath
Top Window Highlight XPath:
1 matching node

```
.//*[@type='submit']//following-sibling::input
```

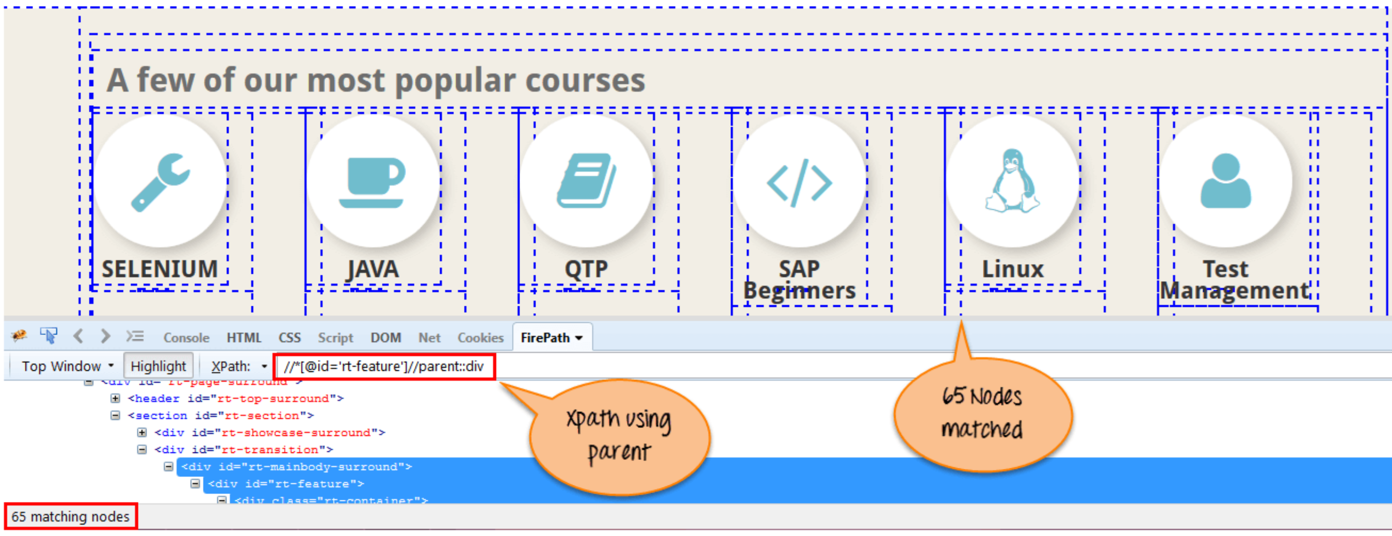
```
<tr>
<tr>
    <td/>
    <td>
        <input type="submit" value="LOGIN" name="btnLogin"/>
        <input type="reset" value="RESET" name="btnReset"/>
    </td>
</tr>
</tbody>
</table>
</form>
```

One input nodes matching by using "following-sibling" axis.

6) Parent:

Selects the parent of the current node as shown in the below screen.

```
Xpath=//*[@id='rt-feature']//parent::div
```


A few of our most popular courses
SELENIUM
JAVA
QTP
SAP Beginners
Linux
Test Management
Console HTML CSS Script DOM Net Cookies FirePath
Top Window Highlight XPath:
Xpath using parent
65 Nodes matched

```
//*[@id='rt-feature']//parent::div
```

```
    <div id="rt-page-surround">
      <header id="rt-top-surround">
      <section id="rt-section">
        <div id="rt-showcase-surround">
        <div id="rt-transition">
          <div id="rt-mainbody-surround">
            <div id="rt-feature">
              <div class="rt-container">
```

```
65 matching nodes
```

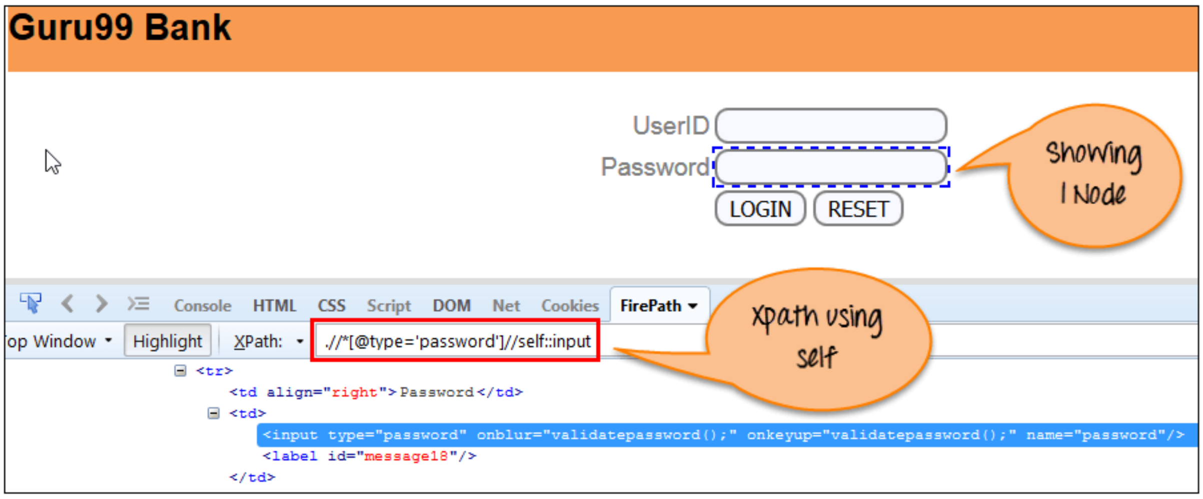
There are 65 “div” nodes matching by using “parent” axis. If you want to focus on any particular element then you can use the below XPath:

Xpath=//*[@id='rt-feature']//parent::div[1]

You can change the XPath according to the requirement by putting [1],[2]…………and so on.

7) Self:

Selects the current node or ‘self’ means it indicates the node itself as shown in the below screen.


Guru99 Bank
UserID
Password
LOGIN
RESET
Showing 1 Node
Console HTML CSS Script DOM Net Cookies FirePath
op Window Highlight XPath:
Xpath using self

```
.//*[@type='password']//self::input
```

```
<tr>
  <td align="right">Password</td>
  <td>
    <input type="password" onblur="validatepassword();" onkeyup="validatepassword();" name="password"/>
    <label id="message18"/>
  </td>
```

One node matching by using “self ” axis. It always finds only one node as it represents self-element.

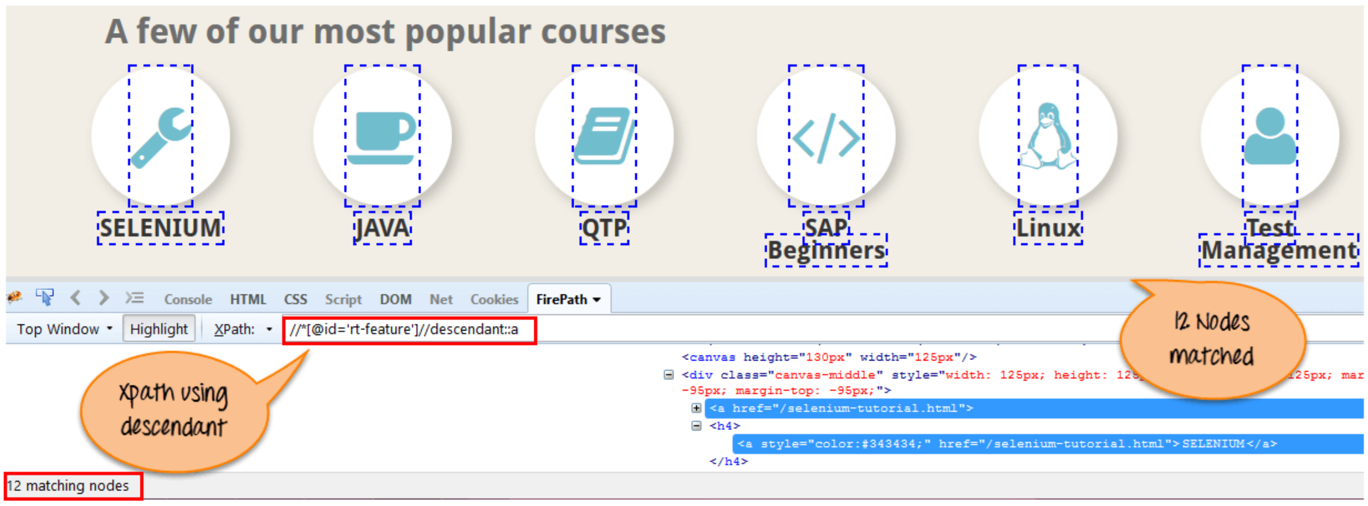
Xpath =//*[@type='password']//self::input

8) Descendant:

Selects the descendants of the current node as shown in the below screen.

In the below expression, it identifies all the element descendants to current element (‘Main body surround’ frame element) which means down under the node (child node , grandchild node, etc.).

Xpath=//*[@id='rt-feature']//descendant::a



//*[@id='rt-feature']//descendant::a

```
<canvas height="130px" width="125px"/>
<div class="canvas-middle" style="width: 125px; height: 12        125px; mar
-95px; margin-top: -95px;">
  <a href="/selenium-tutorial.html">
  <h4>
    <a style="color:#343434;" href="/selenium-tutorial.html">SELENIUM</a>
  </h4>
```

There are 12 “link” nodes matching by using “descendant” axis. If you want to focus on any particular element then you can use the below XPath:

Xpath=//*[@id='rt-feature']//descendant::a[1]

You can change the XPath according to the requirement by putting [1],[2]…………and so on.

Summary:

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 selenium 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, Classname, etc. from the XML document .

What is a 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. The different types of CSS Locator in Selenium IDE

- 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.

tag and id – CSS Selector

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

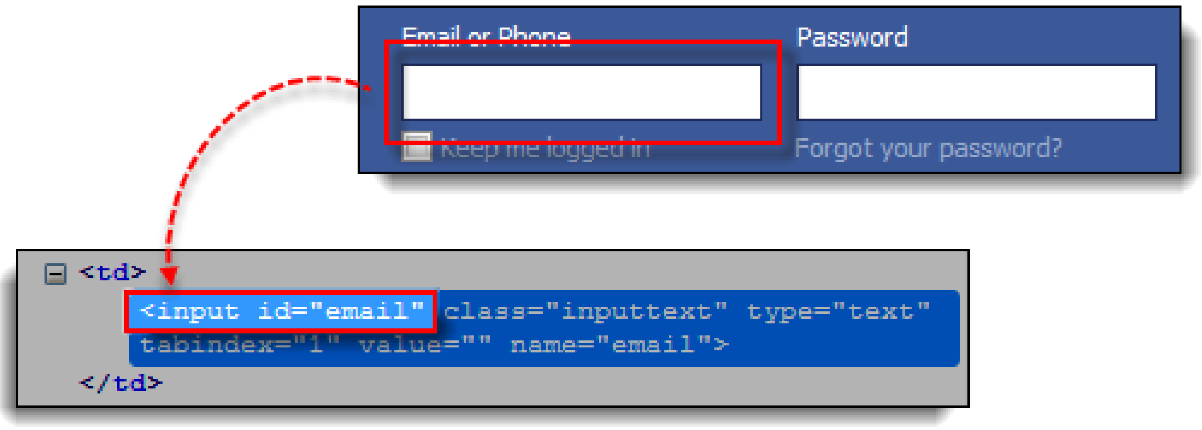
css=*tag#id*

- 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 (#).

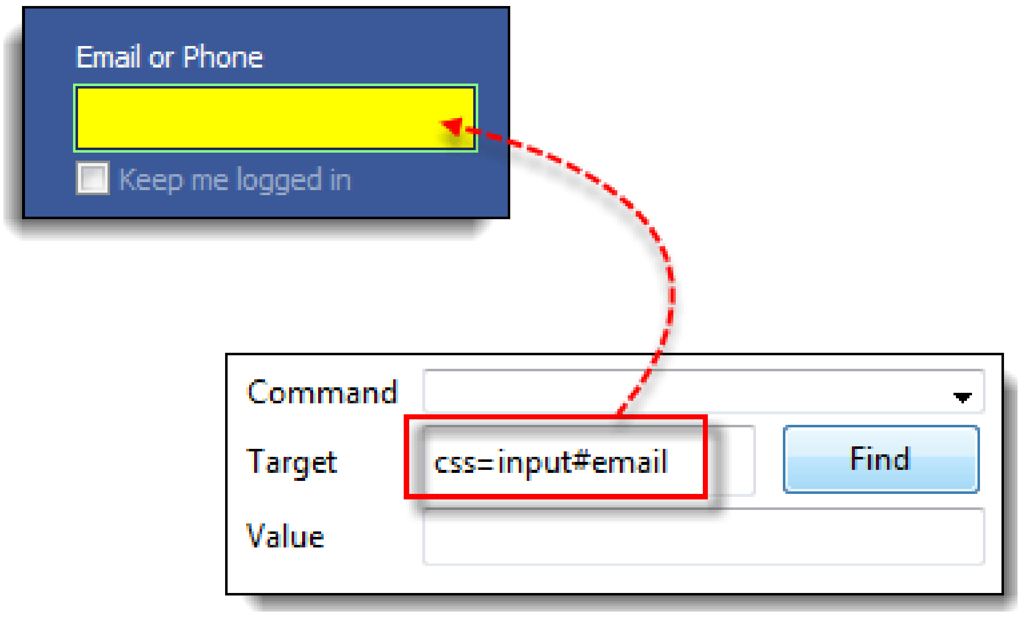
Step 1. Navigate to www.facebook.com. Using Firebug, examine the "Email or Phone" text box.

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



```
<td>
    <input id="email" class="inputtext" type="text"
    tabindex="1" value="" name="email">
</td>
```

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.



Target: css=input#email

tag and class – CSS Selector

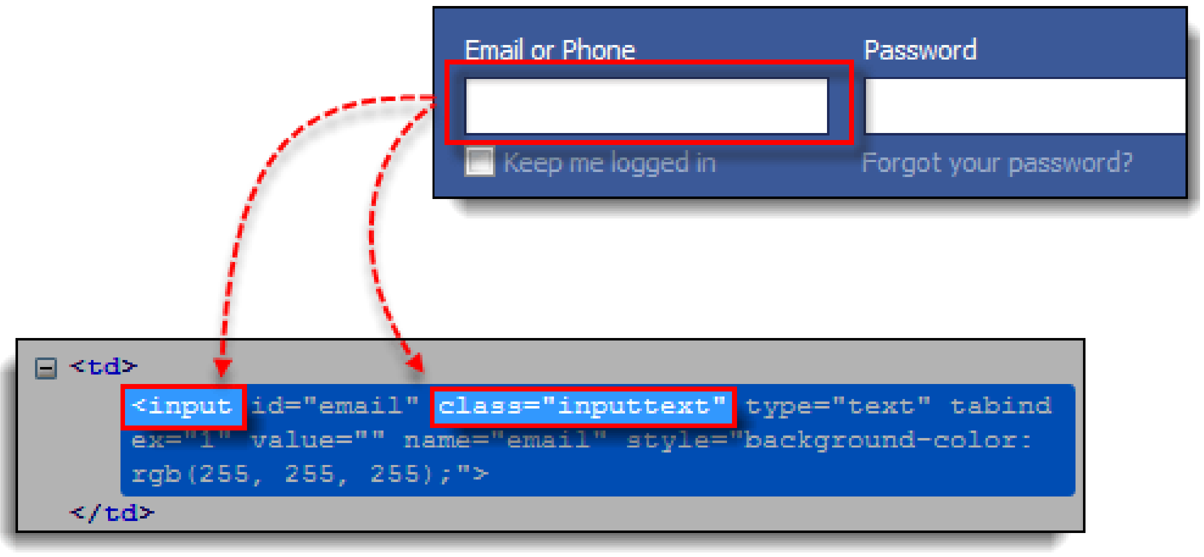
CSS Selector in Selenium using an HTML tag and a class name is similar to using a tag and ID, but in this case, a dot (.) is used instead of a hash sign.

Syntax

css=*tag.class*

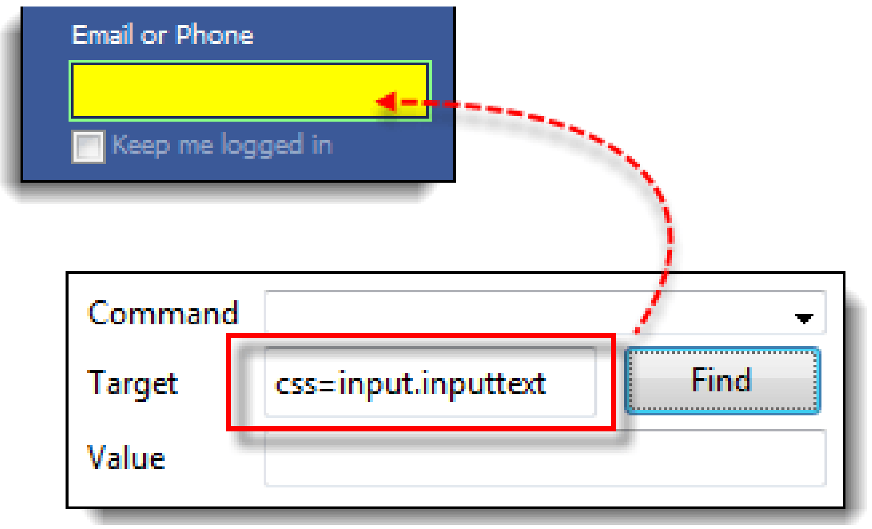
- tag = the HTML tag of the element being accessed
- . = the dot sign. This should always be present when using a CSS Selector with class
- class = the class of the element being accessed

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."



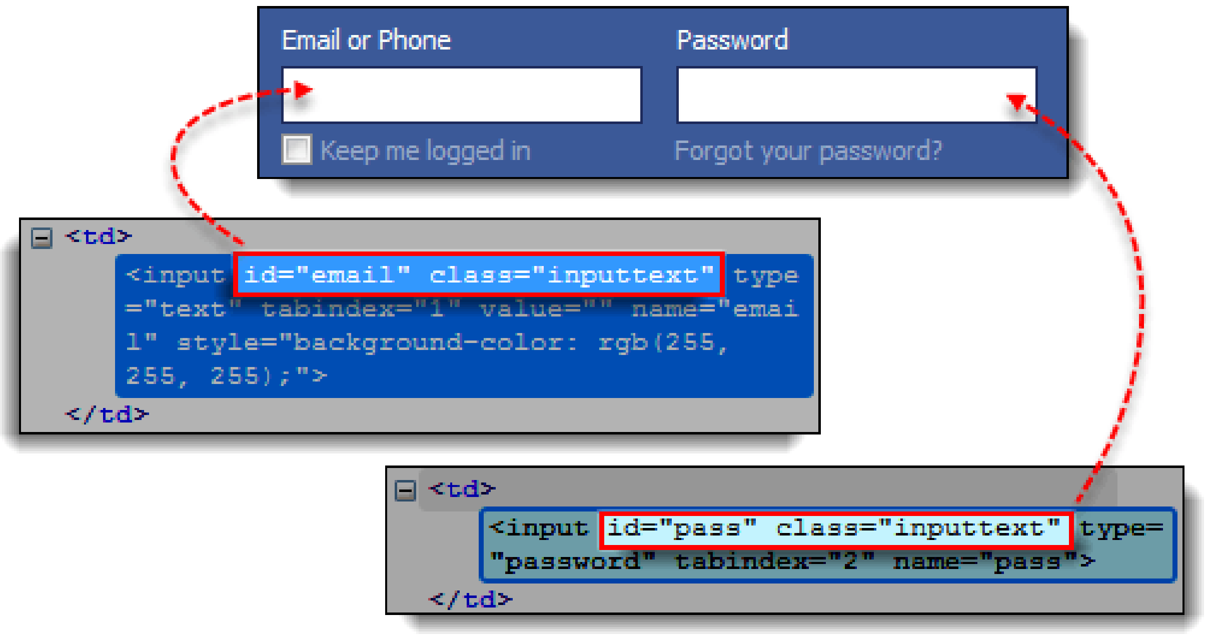
```
<td>
    <input id="email" class="inputtext" type="text" tabindex="1" value="" name="email" style="background-color: rgb(255, 255, 255);">
</td>
```

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.



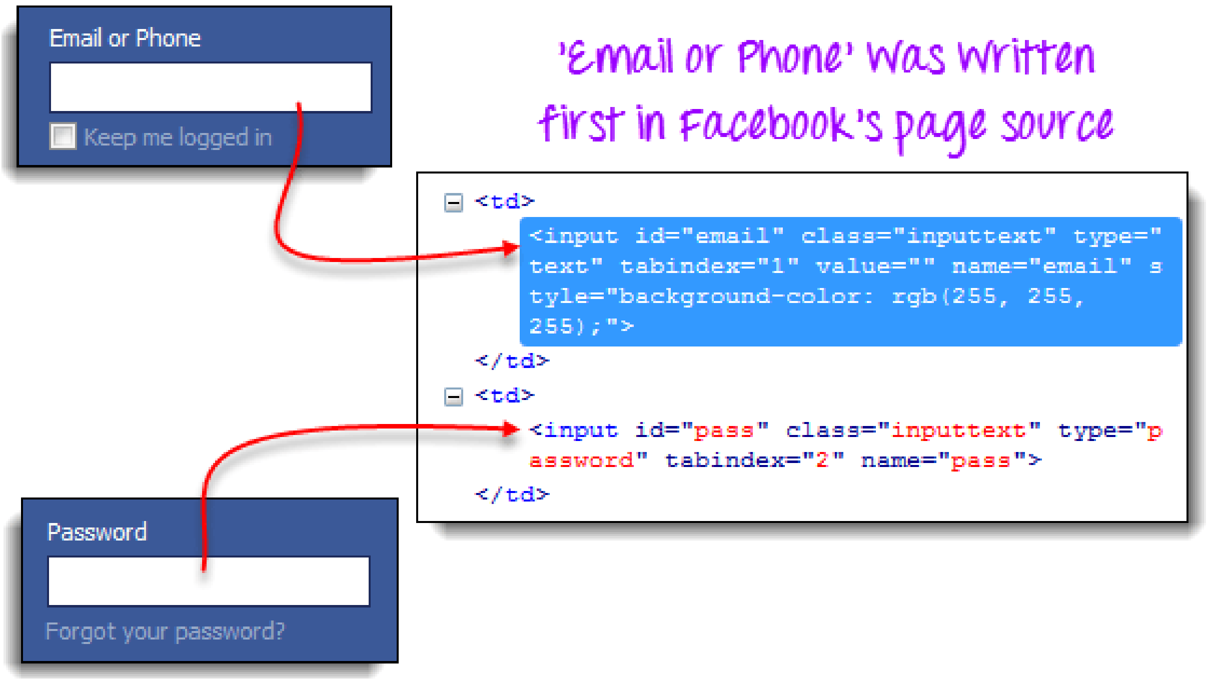
Target: css=input.inputtext

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.



```
<td>
  <input id="email" class="inputtext" type="text" tabindex="1" value="" name="email" style="background-color: rgb(255, 255, 255);">
</td>
<td>
  <input id="pass" class="inputtext" type="password" tabindex="2" name="pass">
</td>
```

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.


Email or Phone
Keep me logged in
'Email or Phone' was written first in Facebook's page source
Password
Forgot your password?

```
<td>
  <input id="email" class="inputtext" type="text" tabindex="1" value="" name="email" style="background-color: rgb(255, 255, 255);">
</td>
<td>
  <input id="pass" class="inputtext" type="password" tabindex="2" name="pass">
</td>
```

tag and attribute – CSS Selector

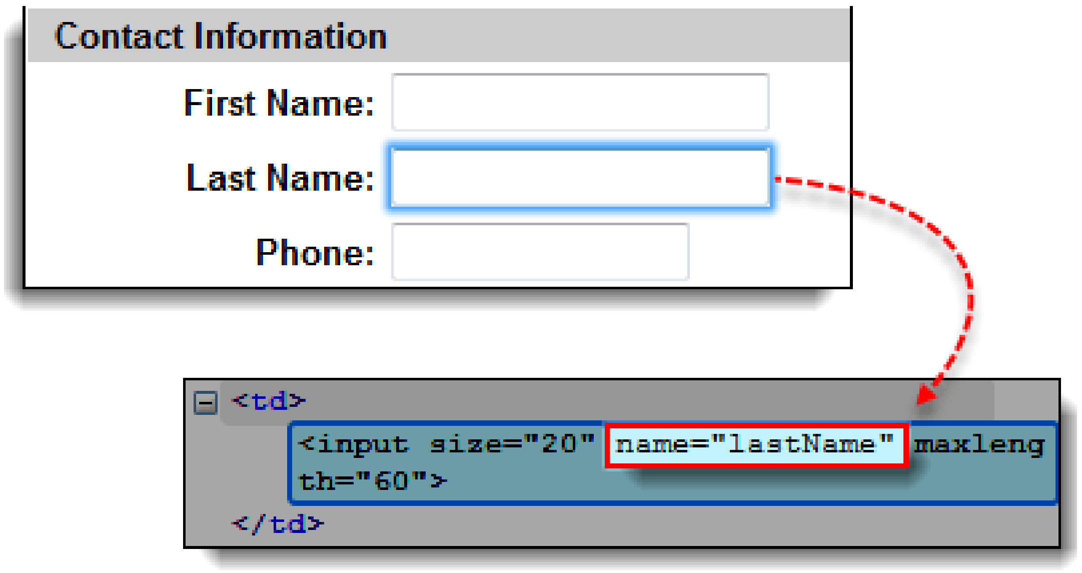
This strategy uses the HTML tag and a specific attribute of the element to be accessed.

Syntax

css=*tag*[*attribute*=*value*]

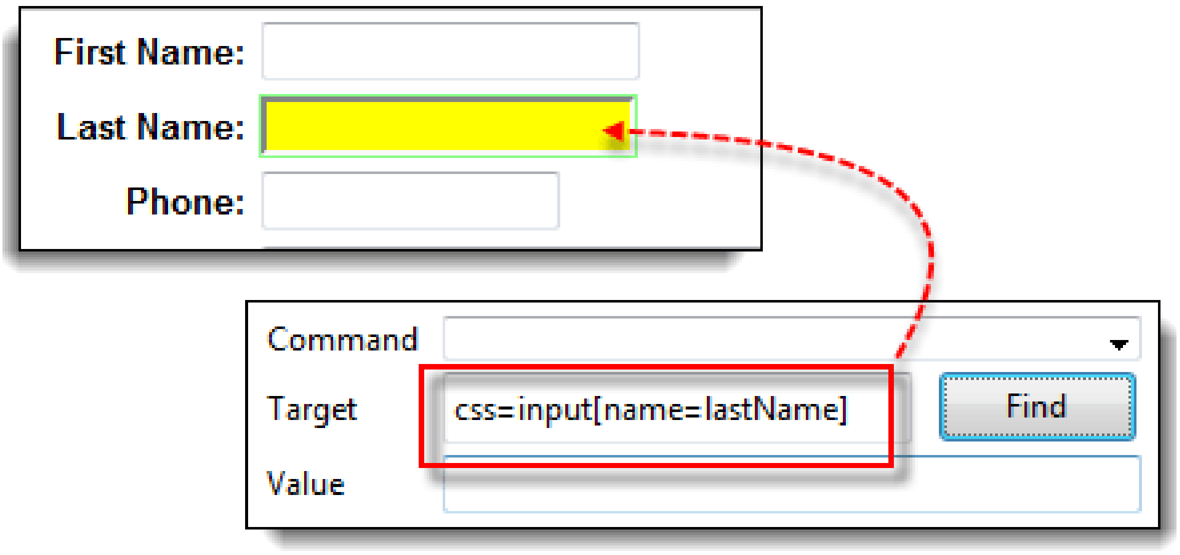
- tag = the HTML tag of the element being accessed
- [and] = square brackets within which a specific attribute and its corresponding value will be placed
- attribute = the attribute to be used. It is advisable to use an attribute that is unique to the element such as a name or ID.
- value = the corresponding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



```
<td>
    <input size="20" name="lastName" maxlength="60">
</td>
```

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.



Target: css=input[name=lastName]

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.

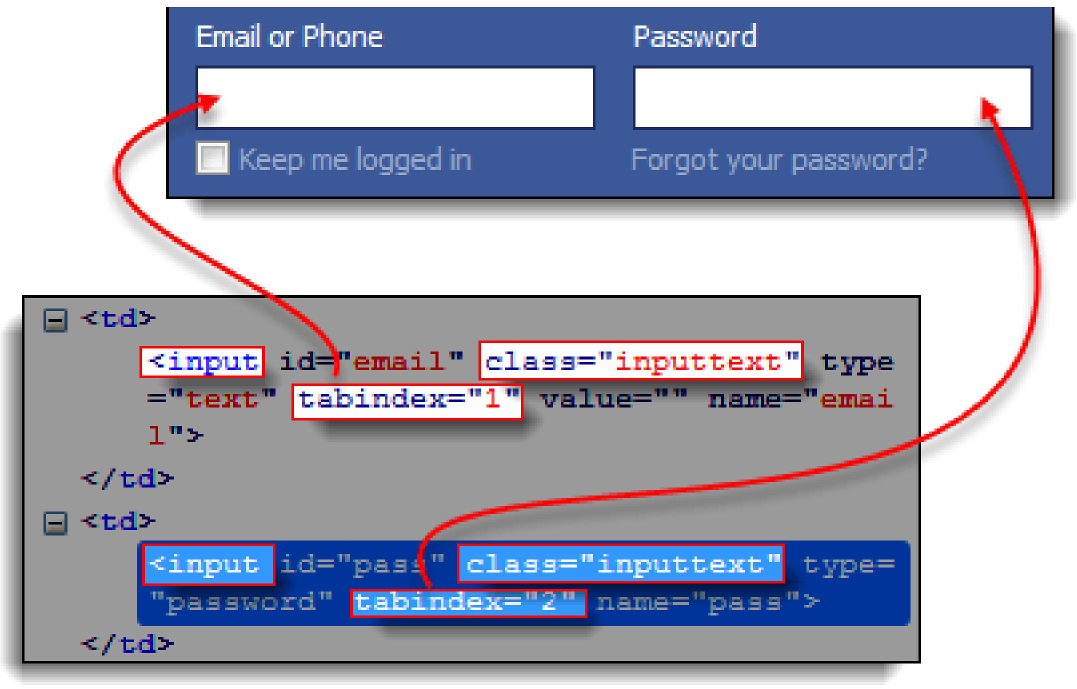
tag, class, and attribute – CSS Selector

Syntax

css=*tag.class*[*attribute=value*]

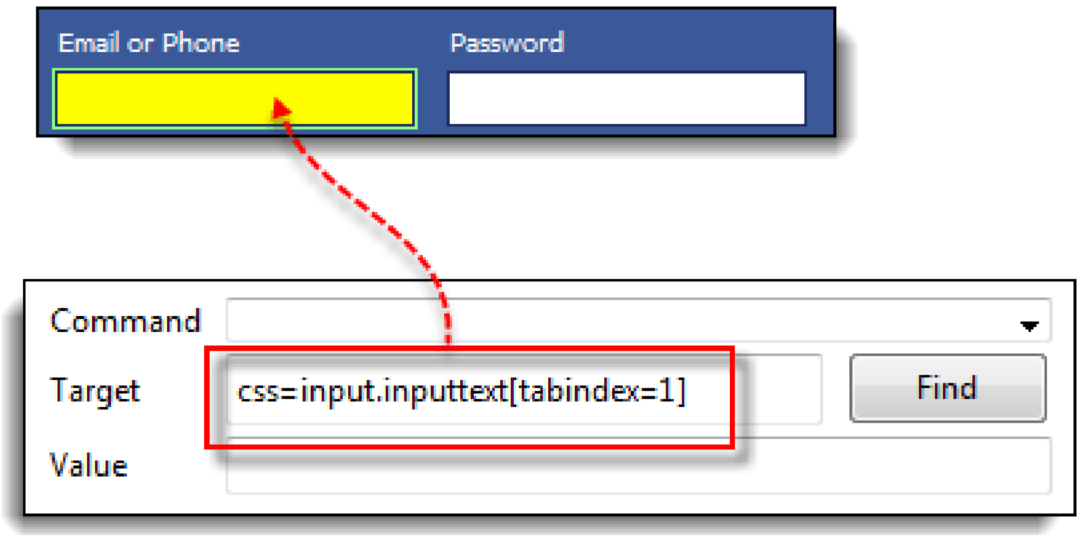
- tag = the HTML tag of the element being accessed
- . = the dot sign. This should always be present when using a CSS Selector with class
- class = the class of the element being accessed
- [and] = square brackets within which a specific attribute and its corresponding value will be placed
- attribute = 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.



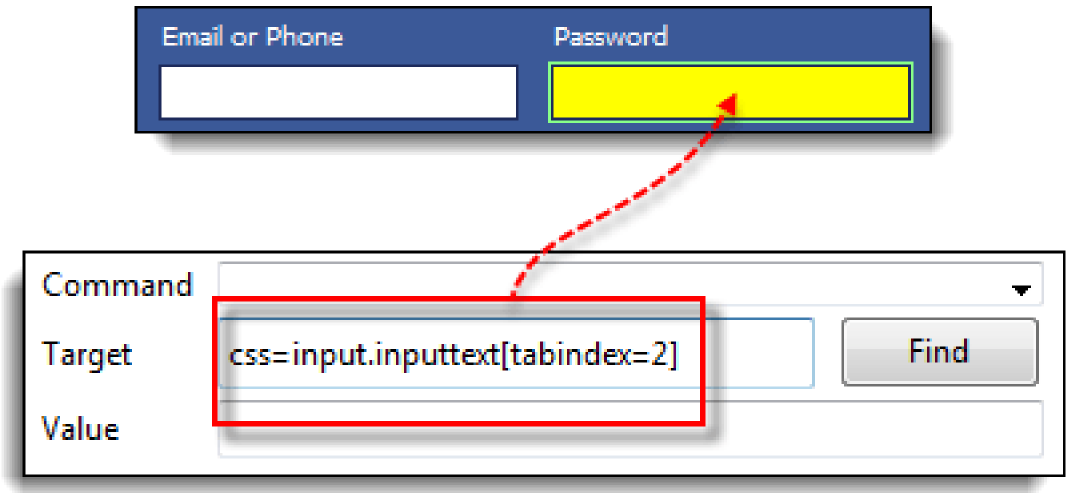
```
<td>
    <input id="email" class="inputtext" type
    ="text" tabindex="1" value="" name="emai
    l">
</td>
<td>
    <input id="pass" class="inputtext" type=
    "password" tabindex="2" name="pass">
</td>
```

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=2]" in Selenium IDE's Target box and click Find. The 'Email or Phone' input box should be highlighted.



css=input.inputtext[tabindex=1]

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.



css=input.inputtext[tabindex=2]

inner text – CSS Selector

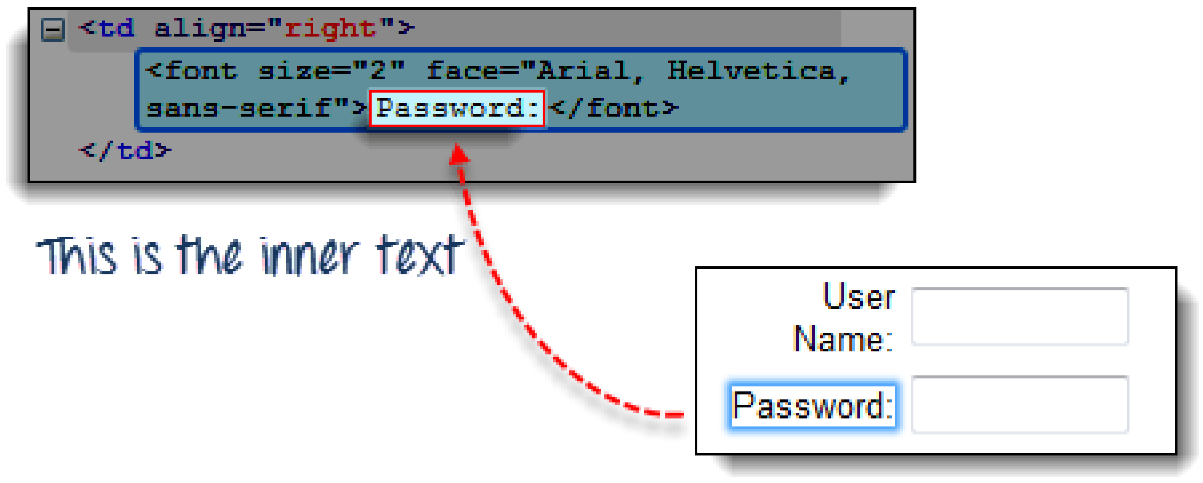
As you may have noticed, HTML labels are seldom given id, name, or class attributes. So, how do we access them? The answer is through the use of their inner texts. **Inner texts are the actual string patterns that the HTML label shows on the page.**

Syntax

css=*tag*:contains("*inner text*")

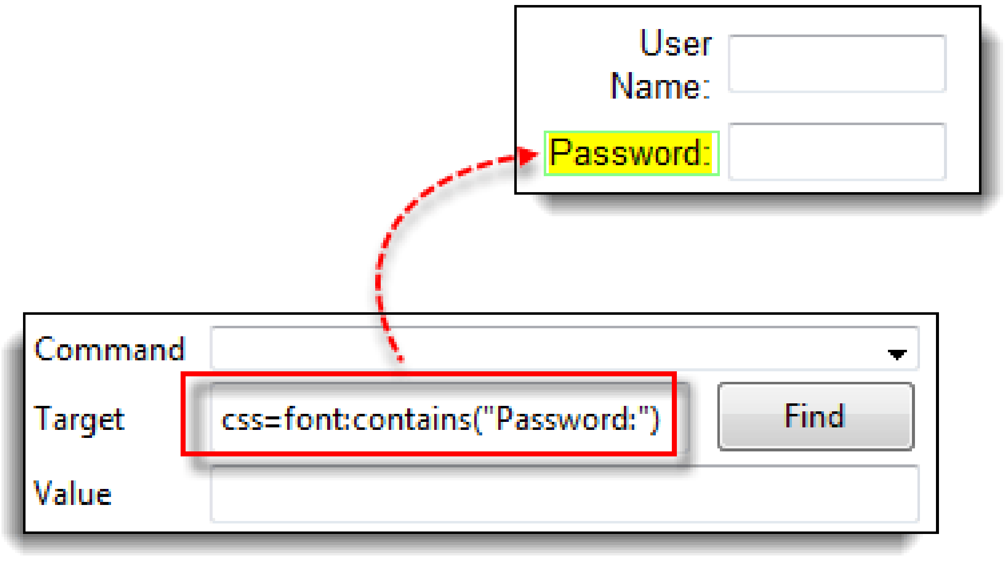
- tag = the HTML tag of the element being accessed
- inner text = the inner text of the element

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.



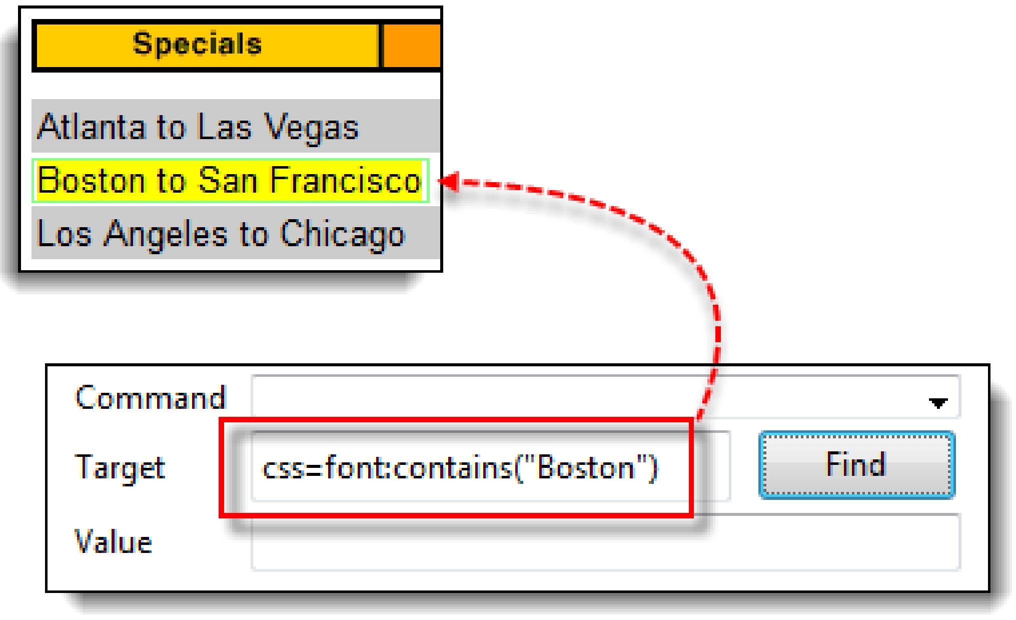
```
<td align="right">
    <font size="2" face="Arial, Helvetica, sans-serif">Password:</font>
</td>
```

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.



Target: css=font:contains("Password:")

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.



css=font:contains("Boston")

Summary

Syntax for Locating by CSS Selector Usage

Method	Target Syntax	Example
Tag and ID	css=*tag*#*id*	css=input#email
Tag and Class	css=*tag*.*class*	css=input.inputtext
Tag and Attribute	css=*tag*[*attribute*=*value*]	css=input[name=lastName]
Tag, Class, and Attribute	css=*tag*.*class*[*attribute*=*value*]	css=input.inputtext[tabindex=1]

Why Do We Need Waits In Selenium?

Most of the web applications are developed using Ajax and Javascript. When a page is loaded by the browser the elements which we want to interact with may load at different time intervals.

Not only it makes this difficult to identify the element but also if the element is not located it will throw an “**ElementNotVisibleException**” exception. Using Selenium Waits, we can resolve this problem.

Let’s consider a scenario where we have to use both implicit and explicit waits in our test. Assume that implicit wait time is set to 20 seconds and explicit wait time is set to 10 seconds.

Suppose we are trying to find an element which has some **"ExpectedConditions** "(Explicit Wait), If the element is not located within the time frame defined by the Explicit wait(10 Seconds), It will use the time frame defined by implicit wait(20 seconds) before throwing an "**ElementNotVisibleException**".

Selenium Web Driver Waits

1. Implicit Wait
2. Explicit Wait

Implicit Wait in Selenium

The **Implicit Wait in Selenium** is used to tell the web driver to wait for a certain amount of time before it throws a "No Such Element Exception". The default setting is 0. Once we set the time, the web driver will wait for the element for that time before throwing an exception.
Selenium Web Driver has borrowed the idea of implicit waits from Watir.

In the below example we have declared an implicit wait with the time frame of 10 seconds. It means that if the element is not located on the web page within that time frame, it will throw an exception.

To declare implicit wait in Selenium WebDriver:

Implicit Wait syntax:

```
driver.manage().timeouts().implicitlyWait(TimeOut, TimeUnit.SECONDS);
```

```
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.annotations.Test;
public class AppTest {

        protected WebDriver driver;

        public void guru99tutorials() throws InterruptedException
        {
        System.setProperty ("webdriver.chrome.driver",".\\chromedriver.exe" );
        driver = new ChromeDriver();

        driver.manage().timeouts().implicitlyWait(10,TimeUnit.SECONDS) ;
        String eTitle = "Demo Guru99 Page";

        String aTitle = "" ;
        // launch Chrome and redirect it to the Base URL

        driver.get("http://demo.guru99.com/test/guru99home/" );

        //Maximizes the browser window

        driver.manage().window().maximize() ;

        //get the actual value of the title
        aTitle = driver.getTitle();
        //compare the actual title with the expected title
        if (aTitle.equals(eTitle))
        {
        System.out.println( "Test Passed") ;
        }
        else {
        System.out.println( "Test Failed" );
        }
        //close browser
        driver.close();
}
}
```

Explicit Wait in Selenium

The **Explicit Wait in Selenium** is used to tell the Web Driver to wait for certain conditions (Expected Conditions) or maximum time exceeded before throwing "ElementNotVisibleException" exception. It is an intelligent kind of wait, but it can be applied only for specified elements. It gives better options than implicit wait as it waits for dynamically loaded Ajax elements.
Once we declare explicit wait we have to use "**ExpectedConditions**" or we can configure how frequently we want to check the condition using **Fluent Wait**.
These days while implementing we are using **Thread.Sleep()** generally it is not recommended to use

In the below example, we are creating reference wait for "WebDriverWait" class and instantiating using "WebDriver" reference, and we are giving a maximum time frame of 20 seconds.

Explicit Wait syntax:

```
WebDriverWait wait = new WebDriverWait(WebDriverRefrence,TimeOut);
```

```
package guru.test99;

import java.util.concurrent.TimeUnit;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;
import org.testng.annotations.Test;

public class AppTest2 {
	protected WebDriver driver;
	@Test
	public void guru99tutorials() throws InterruptedException
	{
	System.setProperty ("webdriver.chrome.driver",".\\chromedriver.exe" );
	driver = new ChromeDriver();
	WebDriverWait wait=new WebDriverWait(driver, 20);
	String eTitle = "Demo Guru99 Page";
	String aTitle = "" ;
	// launch Chrome and redirect it to the Base URL
	driver.get("http://demo.guru99.com/test/guru99home/" );
	//Maximizes the browser window
	driver.manage().window().maximize() ;
	//get the actual value of the title
	aTitle = driver.getTitle();
	//compare the actual title with the expected title
	if (aTitle.contentEquals(eTitle))
	{
	System.out.println( "Test Passed") ;
	}
	else {
	System.out.println( "Test Failed" );
	}
	WebElement guru99seleniumlink;
	guru99seleniumlink=
wait.until(ExpectedConditions.visibilityOfElementLocated(By.xpath(
"/html/body/div[1]/section/div[2]/div/div[1]/div/div[1]/div/div/div/div[2]/div[2]/div/div/div/d
iv/div[1]/div/div/a/i")));
	guru99seleniumlink.click();
	}

}
```

The following are the Expected Conditions that can be used in Selenium Explicit Wait

1. alertIsPresent()
2. elementSelectionStateToBe()
3. elementToBeClickable()
4. elementToBeSelected()
5. frameToBeAvaliableAndSwitchToIt()

6. invisibilityOfTheElementLocated()
7. invisibilityOfElementWithText()
8. presenceOfAllElementsLocatedBy()
9. presenceOfElementLocated()
10. textToBePresentInElement()
11. textToBePresentInElementLocated()
12. textToBePresentInElementValue()
13. titleIs()
14. titleContains()
15. visibilityOf()
16. visibilityOfAllElements()
17. visibilityOfAllElementsLocatedBy()
18. visibilityOfElementLocated()

Fluent Wait in Selenium

The **Fluent Wait in Selenium** is used to define maximum time for the web driver to wait for a condition, as well as the frequency with which we want to check the condition before throwing an "ElementNotVisibleException" exception. It checks for the web element at regular intervals until the object is found or timeout happens.
Frequency: Setting up a repeat cycle with the time frame to verify/check the condition at the regular interval of time

Let's consider a scenario where an element is loaded at different intervals of time. The element might load within 10 seconds, 20 seconds or even more then that if we declare an explicit wait of 20 seconds. It will wait till the specified time before throwing an exception. In such scenarios, the fluent wait is the ideal wait to use as this will try to find the element at different frequency until it finds it or the final timer runs out.

Fluent Wait syntax:

```
Wait wait = new FluentWait(WebDriver reference)
.withTimeout(timeout, SECONDS)
.pollingEvery(timeout, SECONDS)
.ignoring(Exception.class);
```

Above code is deprecated in Selenium v3.11 and above. You need to use

```
Wait wait = new FluentWait(WebDriver reference)
.withTimeout(Duration.ofSeconds(SECONDS))
.pollingEvery(Duration.ofSeconds(SECONDS))
.ignoring(Exception.class);

package guru.test99;

import org.testng.annotations.Test;
import java.util.NoSuchElementException;
import java.util.concurrent.TimeUnit;
import java.util.function.Function;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.FluentWait;
import org.openqa.selenium.support.ui.Wait;
import org.openqa.selenium.support.ui.WebDriverWait;
import org.testng.annotations.Test;

public class AppTest3 {
		protected WebDriver driver;
		@Test
		public void guru99tutorials() throws InterruptedException
		{
		System.setProperty ("webdriver.chrome.driver",".\\chromedriver.exe" );
		String eTitle = "Demo Guru99 Page";
		String aTitle = "" ;
		driver = new ChromeDriver();
		// launch Chrome and redirect it to the Base URL


		driver.get("http://demo.guru99.com/test/guru99home/" );
		//Maximizes the browser window
		driver.manage().window().maximize() ;
		//get the actual value of the title
		aTitle = driver.getTitle();
		//compare the actual title with the expected title
		if (aTitle.contentEquals(eTitle))
		{
		System.out.println( "Test Passed") ;
		}
		else {
		System.out.println( "Test Failed" );
			}

		Wait<WebDriver> wait = new FluentWait<WebDriver>(driver)

				.withTimeout(30, TimeUnit.SECONDS)
				.pollingEvery(5, TimeUnit.SECONDS)
				.ignoring(NoSuchElementException.class);
		WebElement clickseleniumlink = wait.until(new Function<WebDriver, WebElement>(){

			public WebElement apply(WebDriver driver ) {
				return
driver.findElement(By.xpath("/html/body/div[1]/section/div[2]/div/div[1]/div/div[1]/div/div/div/div[2]/div[2]/di
v/div/div/div/div[1]/div/div/a/i"));
			}
		});
		//click on the selenium link
		clickseleniumlink.click();
		//close~ browser
		driver.close() ;
		}
}
```

Explanation of Code

Consider Following Code:

```
Wait<WebDriver> wait = new FluentWait<WebDriver>(driver)

                .withTimeout(30, TimeUnit.SECONDS)
                .pollingEvery(5, TimeUnit.SECONDS)
                .ignoring(NoSuchElementException.class);
```

In the above example, we are declaring a fluent wait with the timeout of 30 seconds and the frequency is set to 5 seconds by ignoring "**NoSuchElementException**"

Consider Following Code:

```
public WebElement apply(WebDriver driver) {
       return
driver.findElement(By.xpath("/html/body/div[1]/section/div[2]/div/div[1]/div/div[1]/div/div/div/div[2]/div[2]/di
v/div/div/div/div[1]/div/div/a/i"));
```

We have created a new function to identify the Web Element on the page. (Ex: Here Web Element is nothing but the Selenium link on the webpage).

Frequency is set to 5 seconds and the maximum time is set to 30 seconds. Thus this means that it will check for the element on the web page at every 5 seconds for the maximum time of 30 seconds. If the element is located within this time frame it will perform the operations else it will throw an "**ElementNotVisibleException**"

Difference Between Implicit Wait Vs Explicit Wait

Following is the main difference between implicit wait and explicit wait in Selenium:

Implicit Wait	Explicit Wait
• Implicit Wait time is applied to all the elements in the script	• Explicit Wait time is applied only to those elements which are intended by us
• In Implicit Wait, we need **not** specify "ExpectedConditions" on the element to be located	• In Explicit Wait, we need to specify "ExpectedConditions" on the element to be located
• It is recommended to use when the elements are located with the time frame specified in Selenium implicit wait	• It is recommended to use when the elements are taking long time to load and also for verifying the property of the element like(visibilityOfElementLocated, elementToBeClickable,elementToBeSelected)