Explanation

As a quality engineer, we all know that for testing a web-based application, we need a perform specific actions(*such as click, type, etc.*) on the *HTML* elements. Now, when we go for automation of those applications, the automation tool should also be capable of performing the same operations on the *HTML* elements that a human is capable of. So, now the question comes, how do these automation tools identify on which *HTML* element they need to perform a particular operation? The answer to this is "*Locators in Selenium.*"

Locators are the way to identify an *HML* element on a web page, and almost all UI automation tools provide the capability to use locators for the identification of *HTML* elements on a web page. Following the same trend, Selenium also possesses the ability to use "*Locators*" for the identification of HTML elements and is popularly known as "*Selenium Locators.*" Selenium supports various kinds of locators. Let's understand the concept of "*Selenium Locators*" in depth by covering the details under the following topics:

What are Locators?

Locator is a command that tells Selenium IDE which GUI elements (say Text Box, Buttons, Check Boxes etc) its needs to operate on. Identification of correct GUI elements is a prerequisite to creating an automation script. But accurate identification of GUI elements is more difficult than it sounds. Sometimes, you end up working with incorrect GUI elements or no elements at all! Hence, Selenium provides a number of Locators to precisely locate a GUI element

Selenium uses what is called locators to find and match the elements of your page that it needs to interact with.

Locators provide a way to access the HTML elements from a web page. In Selenium, we can use locators to perform actions on the text boxes, links, checkboxes and other web elements.

They are the basic building blocks of a web page. A web developer must use a proper and consistent locator scheme for a website. Also, a test engineer must choose the correct locator strategy to automate the online workflows.

However, it gets tough at times to accurately identify a web UI element. And, we end up working with wrong elements or unable to find them.

- *What are locators in Selenium?*
    1. *How to locate a web element in DOM?*
    2. *What locators are supported by Selenium?*
- *How to use locators to find web elements with Selenium?*
    1. *How to locate elements in Selenium using By, ID, Name, Xpath, etc*
    2. *Example showing usage of all the locators?*
- *Best Practices For Using Locators In Selenium*

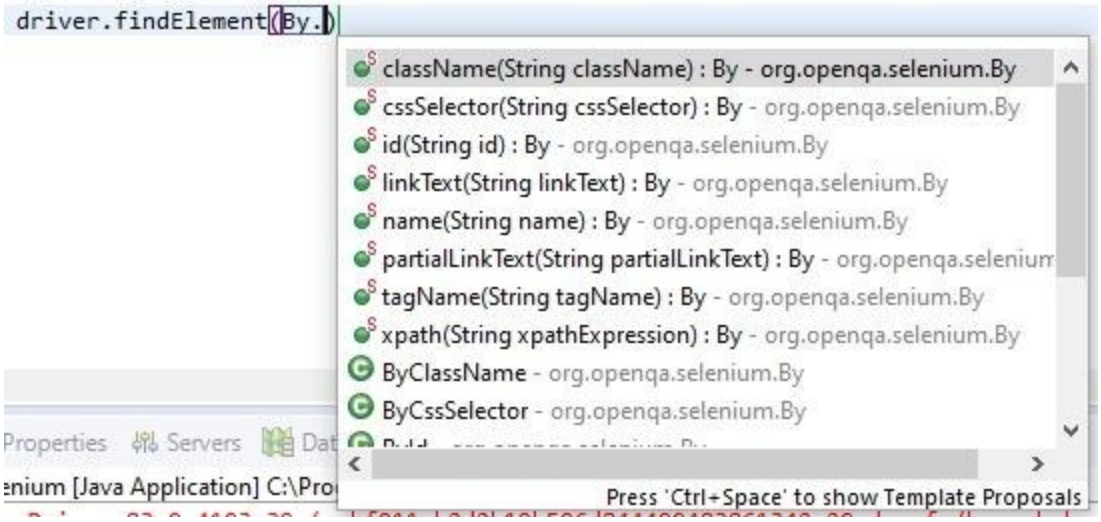
What are Locators in Selenium?

As discussed above, identification of the correct *GUI* element on a web page is pre-requisite for creating any successful automation script. It is where ***locators*** come into the picture. Locators are one of the essential components of Selenium infrastructure, which help Selenium scripts in ***uniquely identifying the WebElements****(such as text box, button, etc.)* present of the web page. So, how do we get the values of these locators? And how to use the same in the automation framework? Let's first understand on the whole page, how we can identify a specific web element in ***DOM*** and then we will try to grab its locator:

What locators are supported by Selenium?

There are various types of locators, using which we can identify a web element uniquely on the Webpage. The following figure shows a good depiction of several types of locators that Selenium supports.

To access all these locators, Selenium provides the "By" class, which helps in locating elements within the *DOM*. It offers several different methods (*some of which are in the image below*) like ***className, cssSelector, id, linkText, name, partialLinkText, taName,*** *and* ***xpath, etc.,*** which can identify the web elements based on their corresponding locator strategies.
You can quickly identify all the supported locators by Selenium by browsing all the visible methods on the "***By"*** class, as shown below:



As we can see, Selenium supports the following locators:

- ***ClassName*** *– A ClassName operator uses a class attribute to identify an object.*
- ***cssSelector*** *– CSS is used to create style rules for webpages and can be used to identify any web element.*
- ***Id*** *– Similar to class, we can also identify elements by using the 'id' attribute.*
- ***linkText*** *– Text used in hyperlinks can also locate element*
- ***name*** *– Name attribute can also identify an element*
- ***partialLinkText*** *– Part of the text in the link can also identify an element*
- ***tagName*** *– We can also use a tag to locate elements*
- ***xpath*** *– Xpath is the language used to query the XML document. The same can uniquely identify the web element on any page.*

Now, let's understand the usage of all these types locators in the Selenium framework:

Locating an Element By ID:
The most efficient way and preferred way to locate an element on a web page is By ID. ID will be the unique on web page which can be easily identified.
IDs are the safest and fastest locator option and should always be the first choice even when there are multiple choices, It is like an Employee Number or Account which will be unique.
Example 1:

```
<div id="toolbar">.....</div>
```

Example 2:

```
<input id="email" class="required" type="text"/>
```

We can write the scripts as

```
WebElement Ele = driver.findElement(By.id("toolbar"));
```

Unfortunately there are many cases where an element does not have a unique id (or the ids are dynamically generated and unpredictable like GWT). In these cases we need to choose an alternative locator strategy, however if possible we should ask development team of the web application to add few ids to a page specifically for (any) automation testing.

Locating an Element By Name:
When there is no Id to use, the next worth seeing if the desired element has a name attribute. But make sure there the name cannot be unique all the times. If there are multiple names, Selenium will always perform action on the first matching element
Example:

```
<input name="register" class="required" type="text"/>
WebElement register= driver.findElement(By.name("register"));
```

Locating an Element By LinkText:
Finding an element with link text is very simple. But make sure, there is only one unique link on the web page. If there are multiple links with the same link text (such as repeated header and footer menu links), in such cases Selenium will perform action on the first matching element with link.

Example:

```
<a href="http://www.seleniumhq.org">Downloads</a>
WebElement download = driver.findElement(By.linkText("Downloads"));
```

Locating an Element By Partial LinkText:
In the same way as LinkText, PartialLinkText also works in the same pattern.

User can provide partial link text to locate the element.
Example:

```
<a href="seleniumhq.org">Download selenium server</a>
WebElement download = driver.findElement(By.PartialLinkText("Download"));
```

Locating an Element By TagName:
TagName can be used with Group elements like , Select and check-boxes / dropdowns.
below is the example code:

```
Select select = new Select(driver.findElement(By.tagName("select")));
select.selectByVisibleText("Nov");
or
select.selectByValue("11");
```

Locating an Element By Class Name:
There may be multiple elements with the same name, if we just use findElementByClassName,m make sure it is only one. If not the you need to extend using the classname and its sub elements.
Example:

```
WebElement classtest =driver.findElement(By.className("sample"));
```

CSS Selector:
CSS mainly used to provide style rules for the web pages and we can use for identifying one or more elements in the web page using css.
If you start using css selectors to identify elements, you will love the speed when compared with XPath.

CSS selectors for Selenium with example

When we don't have an option to choose Id or Name, we should prefer using CSS locators as the best alternative.
CSS is "Cascading Style Sheets" and it is defined to display HTML in structured and colorful styles are applied to webpage.

Selectors are patterns that match against elements in a tree, and as such form one of several technologies that can be used to select nodes in an XML document. Visit to know more W3.Org Css selectors

CSS has more Advantage than Xpath
CSS is much more faster and simpler than the Xpath.
In IE Xpath works very slow, where as Css works faster when compared to Xpath.
Click here to view examples compared with xpath and css

Syntax:

tagName[attributename=attributeValue]
Example 1: input[id=email]
Example 2: input[name=email][type=text]
In CSS there are two special characters which has important role to play.
1. dot(.) refers to class.

Syntax: css=input.submitbtn
For example if the below is the html for a sign button

```
<button class="submit btn primary-btn flex-table-btn js-submit" type="submit" style="background-color:
rgb(85, 172, 238);">
 Log in
 </button>
```

In the above html there are multiple classes used for the single button. How to work in such a situation????
Below are the examples to work with classes. If you observe, we have combined multiple classes to work. As the class is not unique like ID, we may require to join two classes and find the accurate element.

The CSS class selector matches elements based on the contents of their class attribute. In the below exampleprimary-btn is class attribute value.

Example 1: css=.primary-btn
Example 2: css=.btn.primary-btn
Example 3: css=.submit.primary-btn

The above can be written like below in selenium

```
WebElement ele1 = driver.findElement(By.cssSelector(".primary-btn"));
WebElement ele2 = driver.findElement(By.cssSelector(".btn.primary-btn"));
WebElement ele3 = driver.findElement(By.cssSelector(".submit.primary-btn"));
```

Hash(#) refers to Id
Example:

```
css=input[id=email]
```

The above one can be re-written as

```
css=input#destination
```

CSS locator Examples using ID and Class attributes

```
/* below syntax will find "input" tag node which contains "id=email" attribute */

css=input[id=email]
```

In selenium we can write it as

```
WebElement Email = driver.findElement(By.cssSelector("input[id=email]"));
Email.SendKeys("hello@sampleemail.com");
```

You can make use of Selenium IDE to verify if the identifier is working fine or not. If the element has identified, it will highlight the field and html code in Yellow color.

Below syntax will find "input" tag which contains "id=email" and "type=text" attribute. Again here we have added multiple attributes which the input tag has. For username, we will have the text type as 'text' and for password the text type will be 'password'.

```
css=input[name=email][type=text]
```

Below is the syntax for using input Tag and class attribute: It will find input tag which contains "submitButton" class attribute.

```
css=input.rc-button.rc-button-submit
```

Please find the below screen shot with example:
single elements

Using CSS locators, we can also locate elements with sub-strings. Which are really help full when there are dynamically generated ids in webpage

There are there important special characters in css selectors:
1. '^' symbol, represents the starting text in a string.
2. '$' symbol represents the ending text in a string.
3. '*' symbol represents contains text in a string.

CSS Locators for Sub-string matches(Start, end and containing text) in selenium

```
/*It will find input tag which contains 'id' attribute starting with 'ema' text. Email starts with 'ema' */

css=input[id^='ema']
```

If you remove the symbol an try to find the element with same sub-string, it will display error as "locator not found". We can observe the error in the below screen shot. one with error and the other with success

starting substring

/*It will find input tag which contains 'id' attribute ending with 'ail' text. Email ends with 'mail' */

css=input[id$='mail']
/* It will find input tag which contains 'id' attribute containing 'mai' text. Email contains 'mai' */

css=input[id*='mai']
CSS Element locator using child Selectors

/* First it will find Form tag following remaining path to locate child node.*/

css=form>label>input[id=PersistentCookie]
CSS Element locator using adjacent selectors
/* It will try to locate "input" tag where another "input" tag is present on page. the below example will select third input adjacent tag.

css=input + input + input
CSS selector to select first element

We can do this in two ways : -

First using first-of-type - which represents the first element among siblings of its element type and :nth-of-type() matches elements of a given type, based on their position among a group of siblings.

Let us see an example on this using html.

```
<div class="home">
   <span>blah</span>
   <p>first</p>
   <p class="red">second</p>
   <p class="red">third</p>
   <p class="red">fourth</p>
</div>
```

To select the first element with class 'red', css selector should be .red:first-of-type and using nth-of-type, css should be .red:nth-of-type(1)

CSS selector to select last element

Using :last-of-typeselector, we can target the last occurrence of an element within its container. let us look at the below example with html :-

```
<div class="cList">
   <div class="test"> hello test example1 </div>
   <div class="test"> hello test example2 </div>
   <div class="test"> hello test example3 </div>
</div>
```

In the above example, if we want to select ' hello test example3', css selector should be .test:last-of-type.

We can you use Css Selectors to make sure scripts run with the same speed in IE browser. CSS selector is always the best possible way to locate complex elements in the page.

Example:

```
WebElement CheckElements = driver.findElements(By.cssSelector("input[id=email']"));
```

XPath Selector:
XPath is designed to allow the navigation of XML documents, with the purpose of selecting individual elements, attributes, or some other part of an XML document for specific processing

There are two types of xpath

1. Native Xpath, it is like directing the xpath to go in direct way. like
Example:
html/head/body/table/tr/td

Here the advantage of specifying native path is, finding an element is very easy as we are mention the direct path. But if there is any change in the path (if some thing has been added/removed) then that xpath will break.

2. Relative Xpath.
In relative xpath we will provide the relative path, it is like we will tell the xpath to find an element by telling the path in between.
Advantage here is, if at all there is any change in the html that works fine, until unless that particular path has changed. Finding address will be quite difficult as it need to check each and every node to find that path.
Example:
//table/tr/td