Locators for Selenium

Home >> Selenium Tutorials >> Locators for Selenium

Selenium webdriver uses 8 locators to find the elements on web page. The following are the list of object identifier or locators supported by selenium.

We have prioritized the list of locators to be used when scripting.

id *Select element with the specified **@id** attribute.*

Name *Select first element with the specified **@name** attribute.*

Linktext *Select link (anchor tag) element which contains text matching the specified link text*

Partial Linktext *Select link (anchor tag) element which contains text matching the specified partial link text*

Tag Name *Locate Element using a Tag Name .*

Class name *Locate Element using a class Name ..*

Css *Select the element using css selectors.*

Xpath *Locate an element using an XPath expression.*

NOTE : - If you are using selenium latest version, do checkout Relative locators in Selenium version 4.x.x

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.

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 something 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

Example Syntax to work with Image

```
    xpath=//img[@alt='image alt text goes here']
```

Example syntax to work with table

```
    xpath=//table[@id='table1']//tr[4]/td[2]
    xpath=(//table[@class='nice'])//th[text()='headertext']/
```

Example syntax to work with anchor tag

```
    xpath=//a[contains(@href,'href goes here')]
    xpath=//a[contains(@href,'#id1')]/@class
```

Example syntax to work with input tags

```
    xpath=//input[@name='name2' and @value='yes']
```

We will take and sample XML document and we will explain different methods available to locate an element using Xpath