CSS Selectors

This page will show you some CSS rules and pseudo-classes that will help you move your XPATH locators to CSS, a native approach on all browsers.

I: Simple

Direct child

A direct child in XPATH is defined by the use of a "/", while on CSS, it's defined using ">"

Examples:

```
//div/a
css=div > a
```

Child or subchild

If an element could be inside another or one its childs, it's defined in XPATH using "//" and in CSS just by a whitespace.

Examples:

```
//div//a
css=div a
```

Id

An element's id in XPATH is defined using: `"[@id='example']"` and in CSS using: `"#"`

Examples:

```
//div[@id='example']//a
css=div#example a
```

Class

For class, things are pretty similar in XPATH: `"[@class='example']"` while in CSS it's just "."

Examples:

```
//div[@class='example']//a
css=div.example a
```

II: Advanced

Next sibling

This is useful for navigating lists of elements, such as forms or ul items. The next sibling will tell Selenium to find the next adjacent element on the page that's inside the same parent. Let's show an example using a form to select the field after username.

```
</input> </input>
```

Let's write a CSS selector that will choose the input field after "username". This will select the "alias" input, or will select a different element if the form is reordered.

```
css=form input.username + input
```

Attribute values

If you don't care about the ordering of child elements, you can use an attribute selector in Selenium to choose elements based on any attribute value. A good example would be choosing the 'username' element of the form without adding a class.

```
</input> </input> </input> </input>
```

We can easily select the username element without adding a class or an id to the element.

```
css=form input[name='username']
```

We can even add chain filters to be more specific with our selections.

```
css=input[name='continue'][type='button']
```

Here Selenium will act on the input field with name="continue" and type="button"

Choosing a specific match

CSS selectors in Selenium allow us to navigate lists with more finess that the above methods. If we have a ul and we want to select its fourth li element without regard to any other elements, we should use nth-child or nth-of-type.

- `<p>Heading</p>`
- Cat
- Dog
- Car
- Goat

If we want to select the fourth li element (Goat) in this list, we can use the nth-of-type, which will find the fourth li in the list.

```
css=ul#recordlist li:nth-of-type(4)
```

On the other hand, if we want to get the fourth element only if it is a li element, we can use a filtered nth-child which will select (Car) in this case.

```
css=ul#recordlist li:nth-child(4)
```

Note, if you don’t specify a child type for nth-child it will allow you to select the fourth child without regard to type. This may be useful in testing CSS layout in Selenium.

```
css=ul#recordlist *:nth-child(4)
```

Sub-string matches

CSS in Selenium has an interesting feature of allowing partial string matches using ^=, $=, or *=. I’ll define them, then show an example of each:

^= Match a prefix

```
css=a[id^='id_prefix_']
```

A link with an “id” that starts with the text “id_prefix_”

$= Match a suffix

```
css=a[id$='_id_sufix']
```

A link with an “id” that ends with the text “_id_sufix”

*= Match a substring

```
css=a[id*='id_pattern']
```

A link with an “id” that contains the text “id_pattern”

Matching by inner text

And last, one of the more useful pseudo-classes, :contains() will match elements with the desired text block:

```
css=a:contains('Log Out')
```

This will find the log out button on your page no matter where it’s located. This is by far my favorite CSS selector and I find it greatly simplifies a lot of my test code.