

Selenium GUI Automation  
Basics

Version 1.4

Document created by:

Joseph Barber

Document created date:

16/11/2015

Table of Contents

[What does Selenium WebDriver do? 3](#_Toc436643966)

[How to get Selenium WebDriver 3](#_Toc436643967)

[Start developing tests 3](#_Toc436643968)

[Locate HTML Element 4](#_Toc436643969)

[Static Elements 6](#_Toc436643970)

[Dynamic Elements 11](#_Toc436643971)

[Selenium IDE 18](#_Toc436643972)

[Example Program 18](#_Toc436643973)

[Automated Execution 18](#_Toc436643974)

[Helpful Links 18](#_Toc436643975)

[Useful APIs 19](#_Toc436643976)

# What does Selenium WebDriver do?

Selenium WebDriver is used to do web page (GUI) automated testing. Scripts are written in Python and can be executed manually or automatically like a Cron job.

The WebDriver literally drives the website, performing actions on HTML elements. There are 3 simple steps when performing actions on most HTML elements.

1. Locate HTML element
2. Perform action on element

Depending on the HTML element you can perform different actions upon it. I.E a button you would click, but an input field you would send keys.

# How to get Selenium WebDriver

Getting Selenium WebDriver is very simple. A pre-requisite for using this method of getting Selenium is Python-Pip. Follow the install instructions for Selenium WebDriver and Pip if needed.

Python-Pip download: <https://pypi.python.org/pypi/pip>

Selenium WebDriver download: <https://pypi.python.org/pypi/selenium>

# Example Program

**Note:** The example used in this document is not a full program.

In order to view the demo.py example program you must have:

* The WASSP repo installed
* A Titanium Server with working Horizon

The demo file sits at: /wassp-repos/testcases/cgcs/selenium\_tests/Horizon\_Automation/demo.py

To run the demo file type: python2.7 demo.py

# 

# Start developing tests

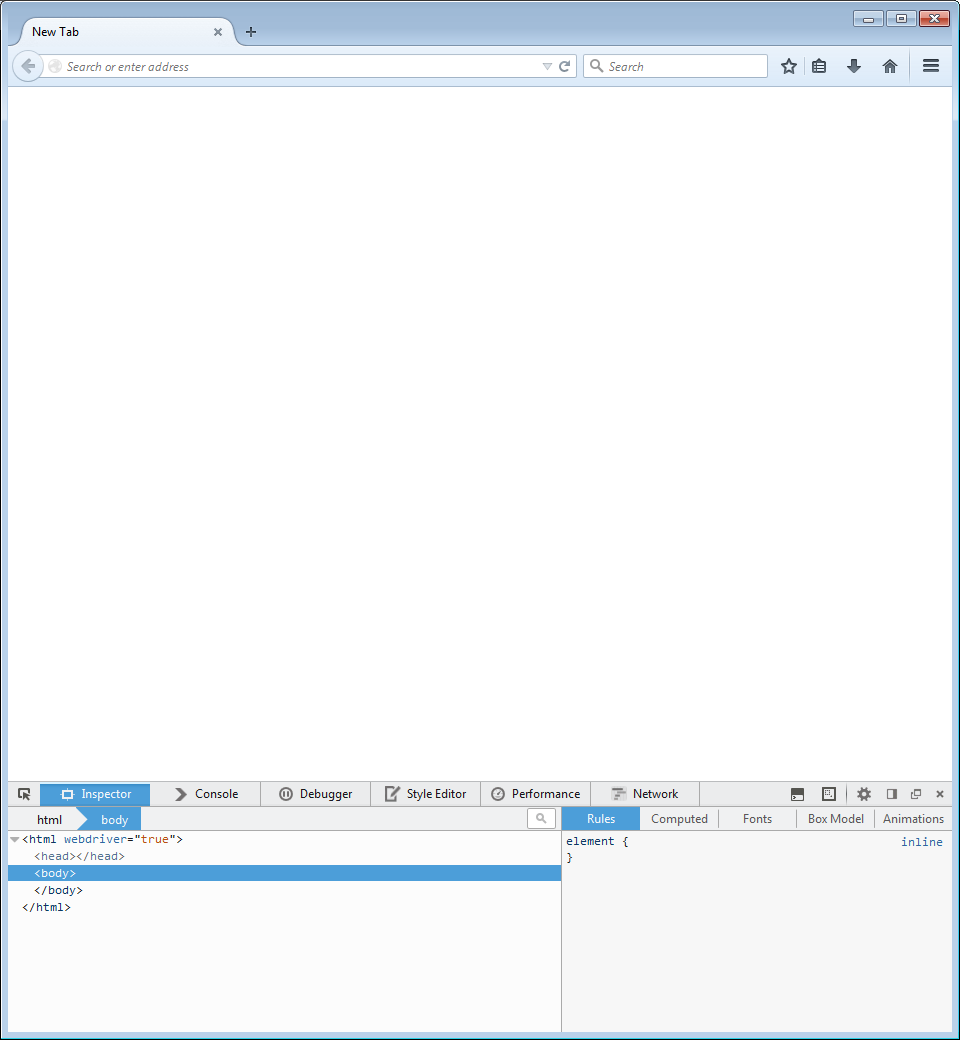
With minimal Python experience, Selenium WebDriver is very simple to use.

As described above, there are 2 steps to using WebDriver.

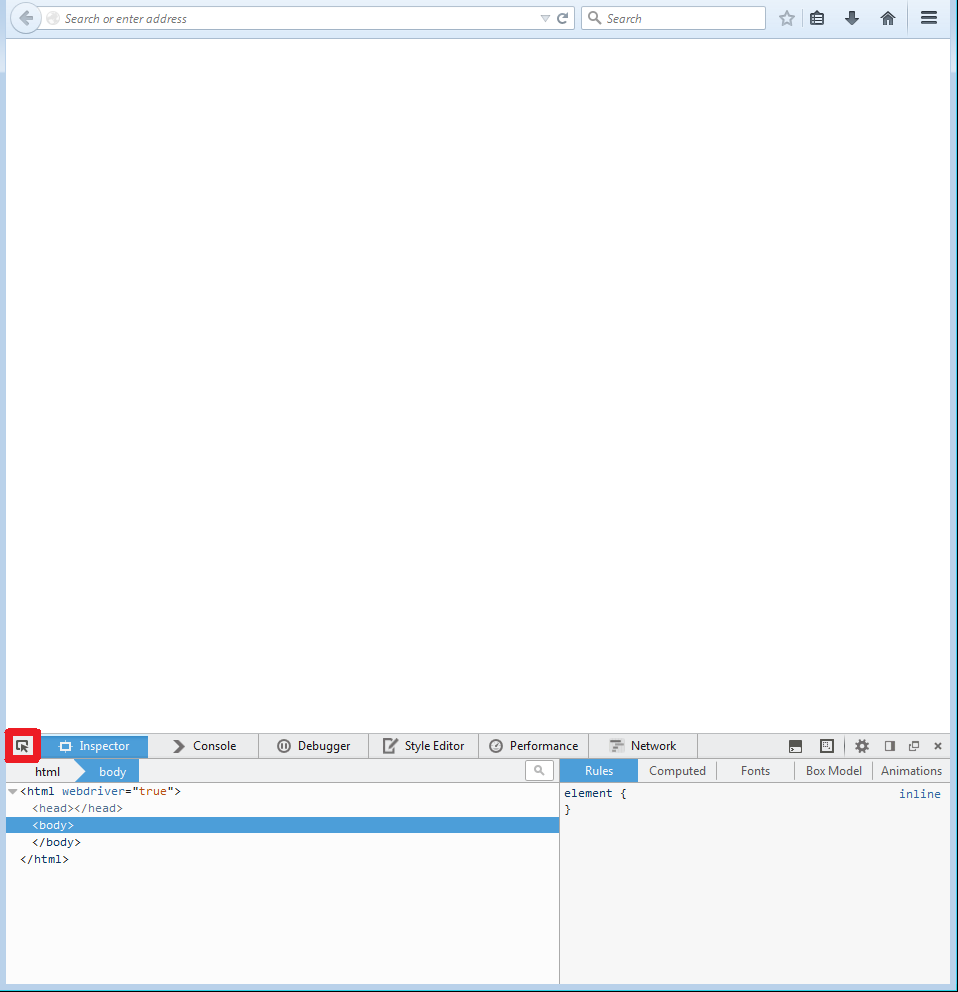
1. Locate the HTML element
2. Perform action on the element from step 1

### Locate HTML Element

In order to locate an HTML element you use your browsers **“Inspector Tool”**. In order to open the browser (in this case Firefox) inspector hit F12 on the keyboard and click Inspector. It will look like this:



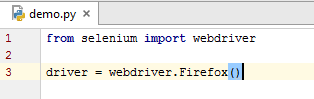
The browser **“Inspector Tool”** is highlighted in the red square below. This tool allows you to click any HTML element and display its properties (id, name, CSS class, etc.). See example on page 7.



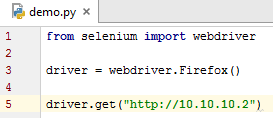
In our example we will be using Horizon’s login page.

### Static Elements

The very first thing we need to do is import the WebDriver and call upon Firefox.



Now we can get our web page.

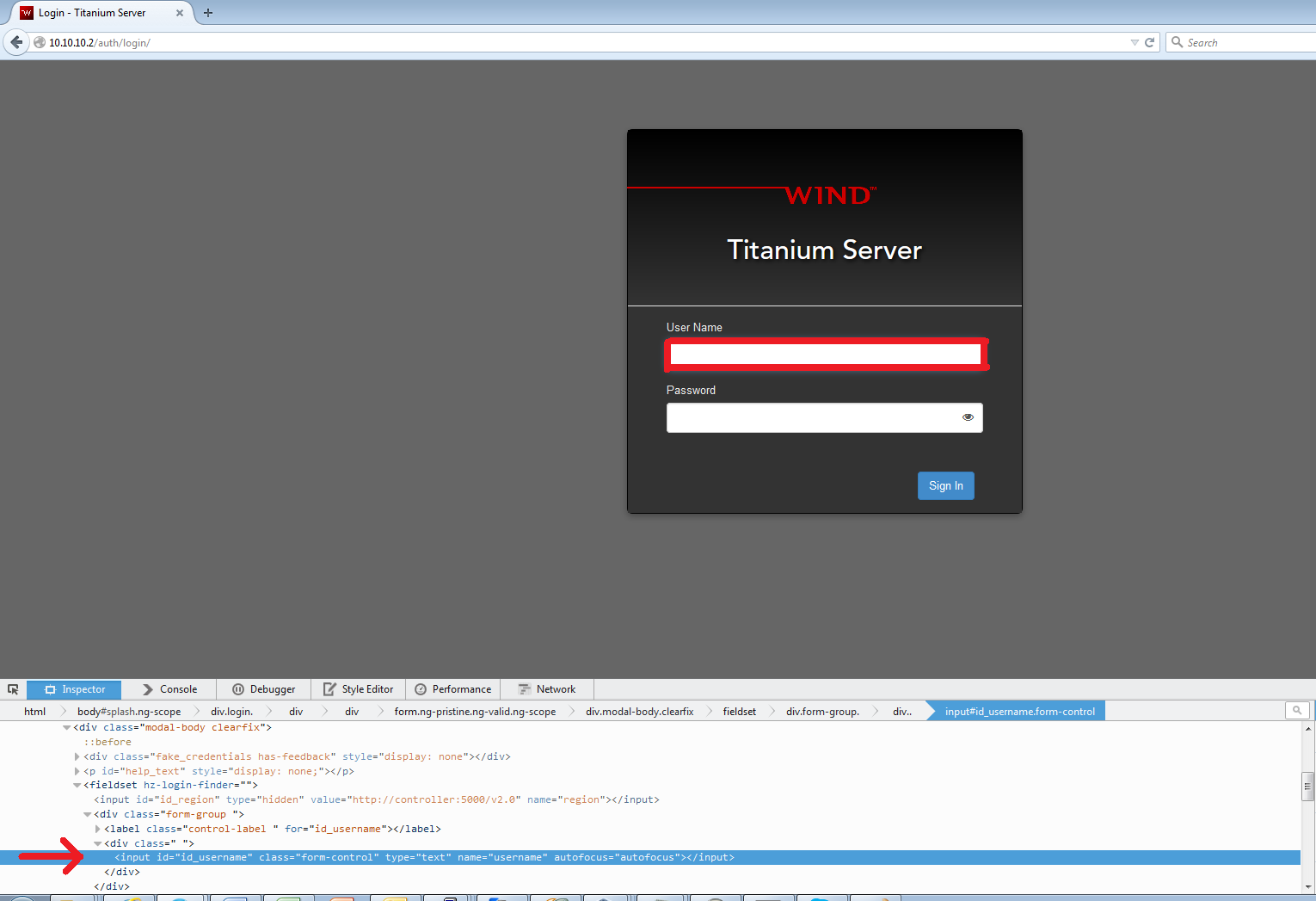


Once we have our web page, we can start to look for HTML elements. There are 3 main ways to looks for HTML elements.

1. **Locate element by id**
2. **Locate element by CSS class**
3. **Locate element by XPath**

## Locate Element by id

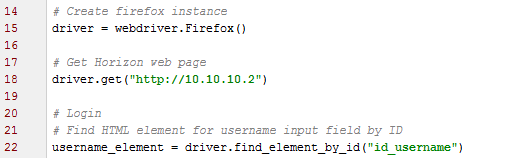
Here we are looking for the username input field id on the web page.



The HTML elements on the Horizon login page is static (they will never change). Meaning they can be located easily using id. The use for CSS class and XPath are more for dynamic HTML elements.

We can find the id for the username field by using the **“Inspector Tool”.**

Now that we have our HTML element we can perform an action on that element found. In our case we found the username input field by id.



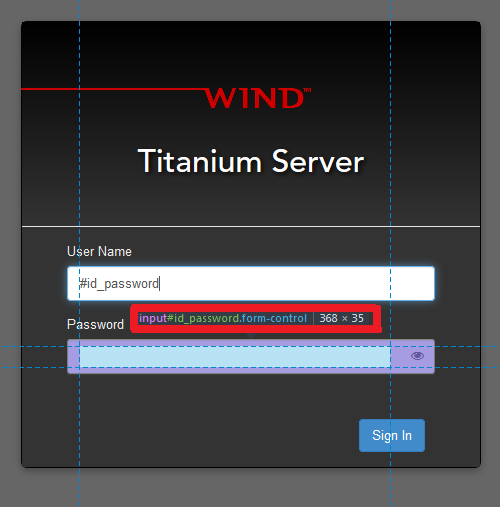
Our action to perform on the input field is to send keys.



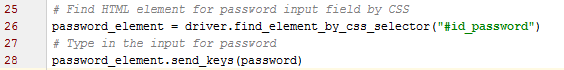
We can now repeat the process for the password input field and the Sign In button. Instead of locating the HTML element by id we will find the password input field by CSS and the ‘Sign In’ button by XPath. In our case we can locate the HTML elements by id only but for example we will use CSS and XPath.

## Locate Element by CSS

Locating the HTML password element by CSS we use the **“Inspector Tool”** and hover over the password input field.

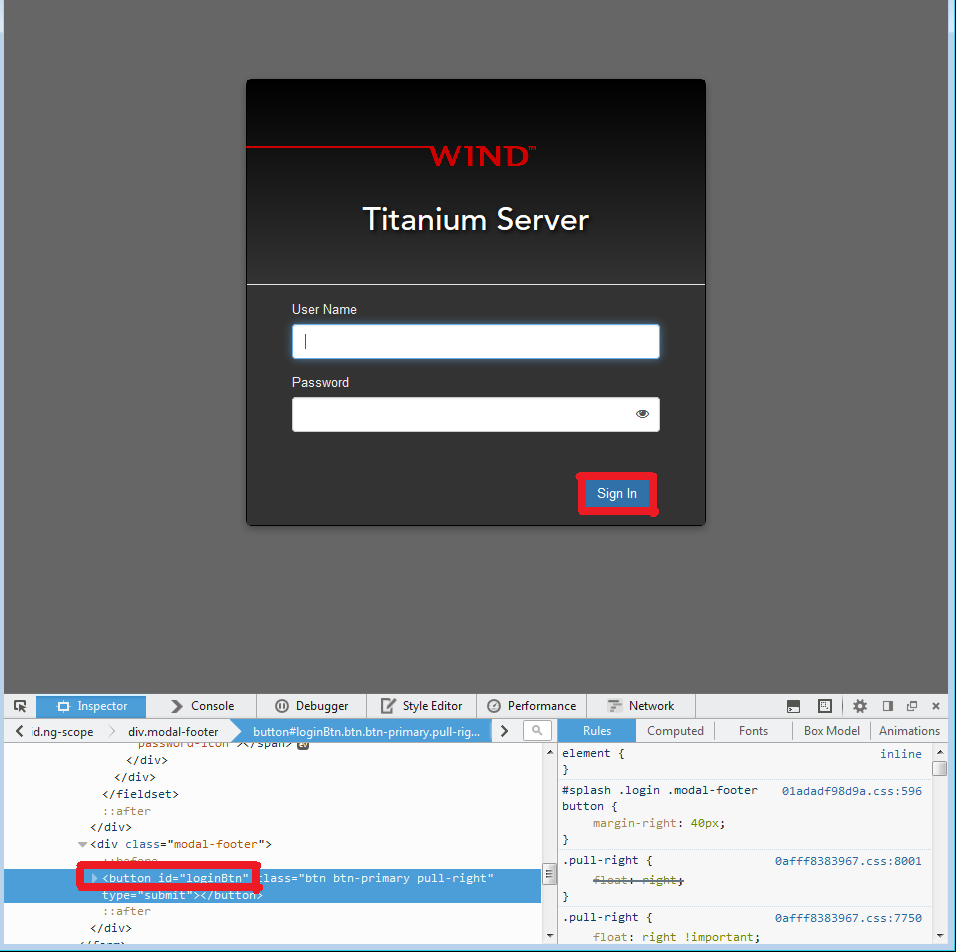


The exact same actions will be performed on password field as the username field above.



## Locate Element by XPath

Locating the HTML ‘Sign In’ button element by XPath we use the **“Inspector Tool”** to click on the ‘Sign In’ button. We can see that the HTML element is a button with the id of “loginBtn”. So we use XPath to search for the button on the entire login web page.



Our ‘Sign In’ HTML element is a button so instead of sending keys, we click it.



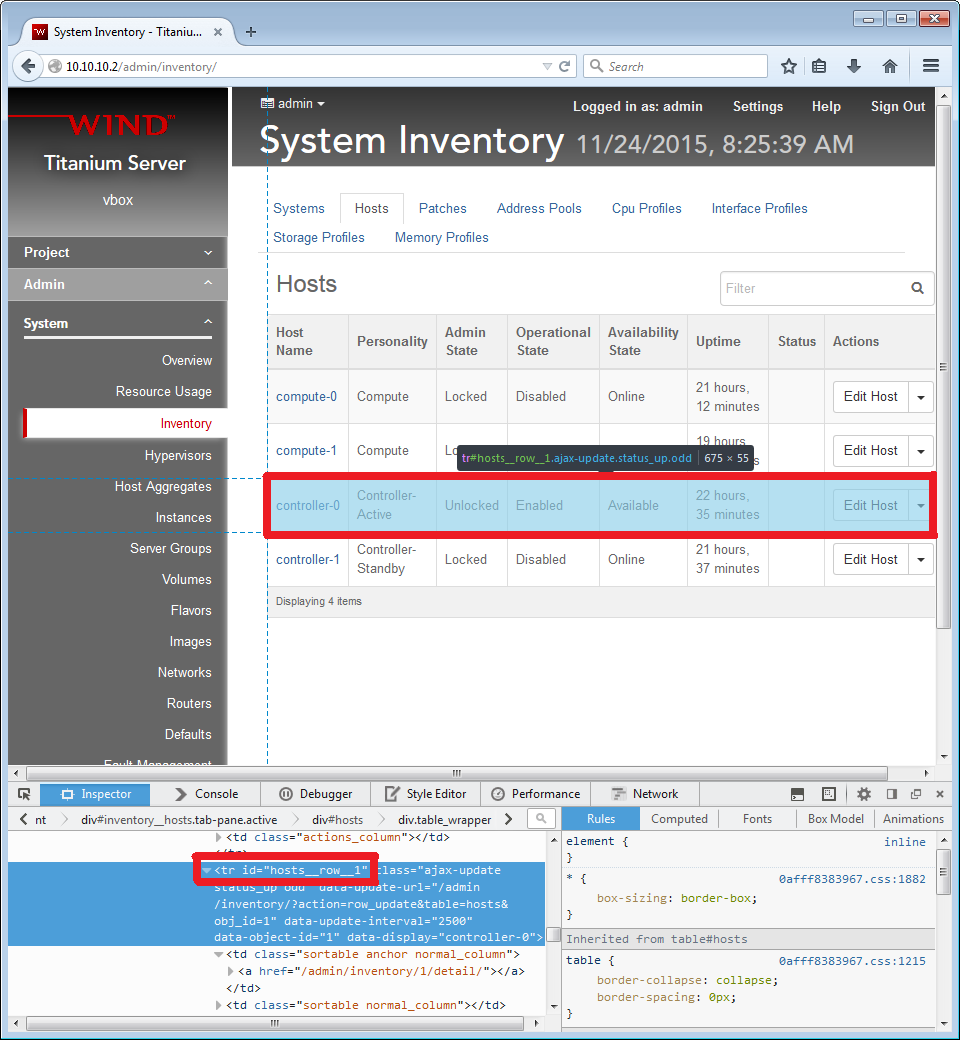
### Dynamic Elements

We talked about static HTML elements like an input field and buttons. However things get a bit complicated when you have to locate and use dynamic elements like columns in a table. (Note: This may not be the best way to find dynamic HTML elements. It’s just the way that I chose to use)

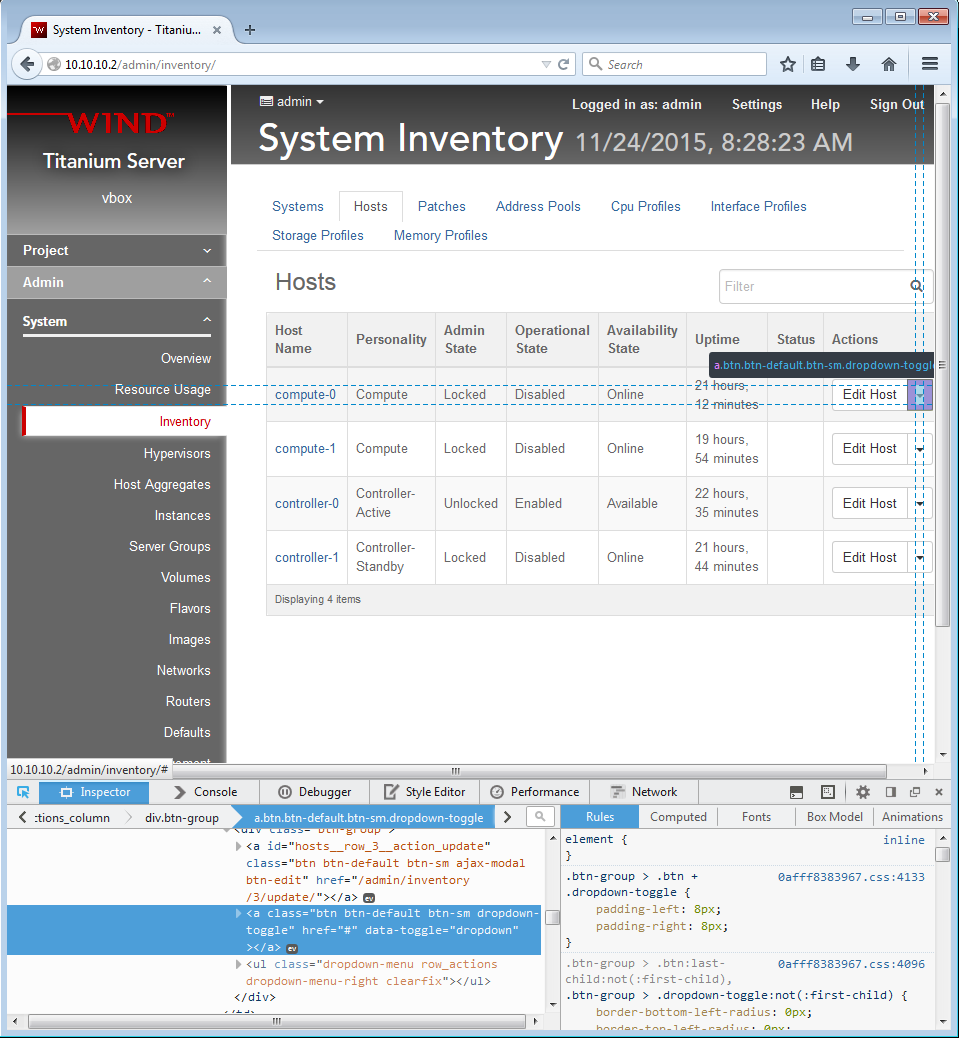
When trying to locate dynamic HTML elements you have 2 main solutions. Locating by id or locating by CSS selector.

The method that I chose to use works on Firefox only. Firefox has another tool called **‘Copy Unique Selector’**. For our example this allows us to get the drop down button in the correct row and column.

Here we have the Inventory in Horizon. We have 4 hosts, and those hosts have different row ids. The row id is dependent on creation time, not on the order in the table. I.E you can’t say that because controller-0 is the 3rd host that it has an id of 3. The id of controller-0 is actually 1 because it was the first host created.



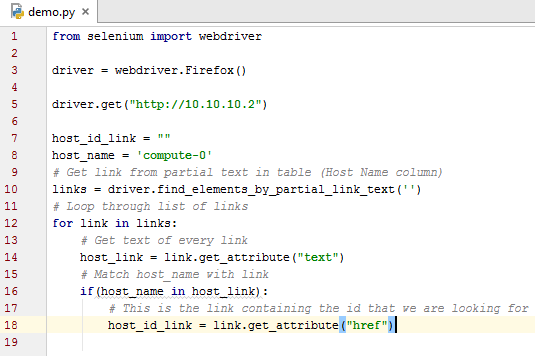
We use the same **“Inspector Tool”** as above to select the drop down for the HTML element that we want, in our case compute-0’s drop down.



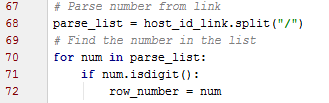
Now instead of assuming that compute-0 has an id of 1, we will get its id through the HTML link from the column ‘Host Name’ and then complete the id location string.

Example of HTML link for compute-0: <http://10.10.10.2/admin/inventory/3/detail/>

The code below is looping through all the links on the inventory page looking to match the host name provided.

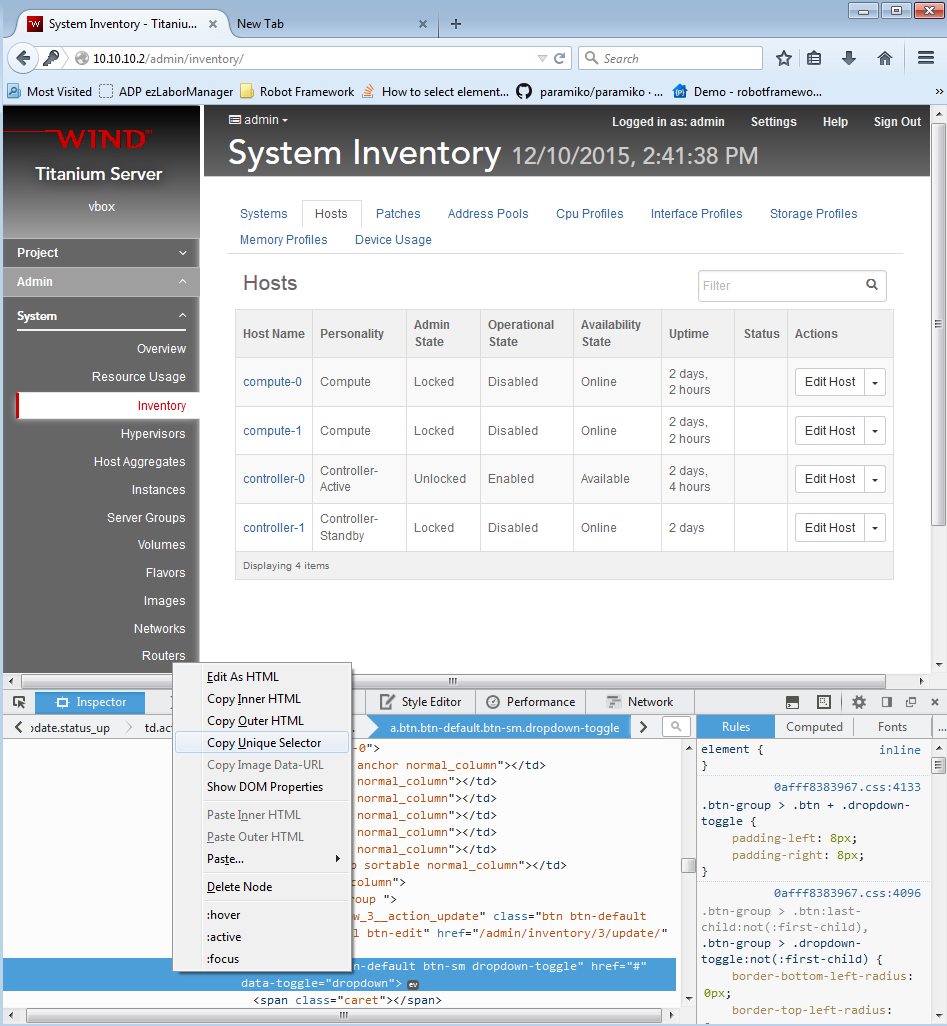


We parse the HTML link found and the only number in the link is our host id (in this case 3). Now that we have the host id, we can move on to creating the location string and finally clicking the drop down.



To create the id location string we need to use the **‘Copy Unique Selector’** tool and cut out the id, making this line dynamic for any id to be used. In this way you can pass the id from the HTML link for the host into the id location string. (For compute-0 the id is 3). See example below.

To get the **‘Copy Unique Selector’** use the **“Inspector Tool”** to select which HTML element you need. Then by right clicking on the HTML code as shown below, you have the option to **‘Copy Unique Selector’**. This now copies the string to your clipboard.



The string that was copied to the clipboard from the above example using the **‘Copy Unique Selector’** tool is:

#hosts\_\_row\_\_3 > td:nth-child(8) > div:nth-child(1) > a:nth-child(2)

We cut the above string at the id, which is 3, and remove the id allowing us to concatenate any id inside the string (shown below). I made constants for the first half of the string and the second half.



I append the found id to the first half and then append the second half to the id. I created a constant for reusability and clarity.



The process from here on is the exact same as static, you locate the element, and then click the element.



# Selenium IDE

Selenium makes an IDE that allows you to record your keystrokes and generates Python code for you. There are a few problems for this and the biggest problem is being modular. Code reuse is important when creating/adding onto snippets of code.

The good thing about the Selenium IDE is that for a general test case, the IDE will record your steps exactly. When the code is generated you can go in and modify the code as needed.

One last point when using Selenium IDE is if the site changes at all the test case is invalid. With creating modules instead of recording a specific test case, it can cover almost all possible solutions.

Link to the IDE is here: <http://www.seleniumhq.org/download/> and scroll down to Selenium IDE download.

# Helpful Links

Reminder when googling for help. Google with Python as a keyword in the search as Selenium WebDriver can be programmed in a few other languages like Java.

<http://www.seleniumhq.org/docs/03_webdriver.jsp> Select Python at the top of the page

Stack Overflow is your best friend when googling.

# Useful APIs

common\_utils.py - Handles common utilities for automating Selenium  
File location: /wassp-repos/testcases/cgcs/selenium\_tests/Horizon\_Automation/common\_utils.py



hosts.py - Handles all host functions  
File location: /wassp-repos/testcases/cgcs/selenium\_tests/Horizon\_Automation/hosts.py

