Python Selenium Tutorial

Created by: Brendan McCann

Page 1

Notes from the Author

This tutorial is meant to be a helpful tool regardless of skill level. I discovered a lot of these things online but I thought it would be helpful to place it all into one place. Note that this tutorial is solely code based and does not explain how to setup Selenium, Python, or Eclipse (those instructions are located in another document). Rather, this tutorial contains code on how to use Selenium driven by Python and uses Eclipse as the programming platform. It does assume a certain level of preexisting knowledge when it comes to Python. All of the code written here is used to test a KX3 device. I have done my best to write out instructions for any code that I have worked with. In order to provide a more comprehensive tutorial I have separated this document into multiple Sections. Section I is about writing code from scratch without any kind of reference files. Section II shows code which implements the Utility classes. Section III shows code which implements the Page Object classes.

Disclaimer: all of the coding examples in this tutorial have been tested and work. However, copying them and pasting them into Eclipse may not work for several reasons. If file structure differs then import statements may not work as intended. In addition Python is very strict about indentation so edits to tabs may be required.

Page 2

Table of Contents

Section I: Starting from Scratch

Section 1.1: Summary…………………………………………........pg. 5

Section 1.2: Initialization/Setup………………………………….....pg. 6

Section 1.3: Running/Closing………………………………………pg. 7

Section 1.4: Test Method................................................................pg. 8-9

Section 1.5: Navigating menus....................................................... pg. 10-11

Section 1.6: Interacting with Checkboxes......................................pg. 12

Section 1.7: Interacting with Select boxes.....................................pg. 13

Section II: Utility Methods

Section 2.1: Summary.....................................................................pg. 14

Section 2.2: ClickOn.......................................................................pg. 14

Section 2.3: DriverControl..............................................................pg. 15

Section 2.4: FileRead........................................................................pg. 16

Section 2.5: FileSetup.......................................................................pg. 16

Section 2.6: Input..............................................................................pg. 17

Section 2.7: Login.............................................................................pg. 18

Section 2.8: Navigate........................................................................pg. 19

Section 2.9: Strong Passwords..........................................................pg. 20

Section 2.10: User Management.......................................................pg. 21-22

Section 2.11: Examples....................................................................pg. 23-24

Section III: Page Objects

Section 3.1: Summary.......................................................................pg. 25

Section 3.2: Naming Convention......................................................pg. 25

Section 3.3: Buttons..........................................................................pg. 25

Section 3.4: Checkboxes...................................................................pg. 26

Section 3.5: Textboxes......................................................................pg. 26

Section 3.6: Select boxes...................................................................pg. 27

Section 3.7: Links..............................................................................pg. 27

Section 3.8: Data...............................................................................pg. 28

Section 3.9: Sort................................................................................pg. 28

Section 3.10: Menus..........................................................................pg. 29

Section 3.11: Examples.....................................................................pg. 29-30

Section IV: File Usage

Section 4.1: Summary.......................................................................pg. 31

Section 4.2: FileRead........................................................................pg. 31

Section 4.3: FileSetup.......................................................................pg. 32-33

Page 3

Section V: Batch Testing

Section 5.1: Summary......................................................................pg. 34

Section 5.2: BatchImports................................................................pg. 34

Section 5.3: BatchSetup...................................................................pg. 35

Section VI: SVN Instructions

Section 6.1: Summary......................................................................pg. 36

Section 6.2: Installing SVN.............................................................pg. 36

Section 6.3: Accessing Repositories................................................pg. 36

Section 6.4: Synchronizing Repositories.........................................pg. 37

Section 6.5: Changing Repositories.................................................pg. 37

Section 6.6: Viewing History...........................................................pg. 38

Appendix

Appendix A: Reference/ Helpful Links............................................pg. 39

Appendix B: Example List...............................................................pg. 39-40

Appendix C: Coding Template.........................................................pg. 41

Page 4

Section I:

Starting from Scratch

Section 1.1 (Summary):

Section I is devoted to explaining the Selenium Python code in its base form. The code here does not implement the methods from PageObjects or Utilities.

Example 1.1: represents a complete test case; we will be breaking it up in sections (1.2-1.4) and examining it in pieces. The code will attempt to log into the KX3 system.

Import unittest

from selenium import webdriver

from selenium.webdriver.support.ui import WebDriverWait

from selenium.webdriver.common.by import By

from selenium.webdriver.support import expected\_conditions as EC

from selenium.webdriver.common.keys import Keys

class **TestFromScratch**(unittest.TestCase):

def **etup**(*self*):

*self*.driver = webdriver.Firefox()

def **test\_login**(*self*):

driver = *self*.driver

driver.get(*“https://192.168.51.46”*)

wait = WebDriverWait(driver, 10)

element= wait.until(EC.presence\_of\_element\_located((By.NAME, *‘login’*)))

element.clear()

element.send\_keys(*‘admin’*)

element= wait.until(EC.presence\_of\_element\_located((By.NAME, *‘password’*)))

element.clear()

element.send\_keys(*‘raritan0’*)

element= wait.until(EC.element\_to\_be\_clickable((By.NAME, *‘action\_login’*)))

element.send\_keys(Keys.RETURN)

def **etupwn**(*self*):

*self*.driver.close()

if \_\_name\_\_ == *“\_\_main\_\_”*:

unittest.main()

Page 5

Section 1.2 (Initialization/Setup):

Example 1.2: shows the basic code we need to setup our class and driver.

Import unittest

from selenium import webdriver

from selenium.webdriver.support.ui import WebDriverWait

from selenium.webdriver.common.by import By

from selenium.webdriver.support import expected\_conditions as EC

from selenium.webdriver.common.keys import Keys

class **TestFromScrtach**(unittest.TestCase):

def **etup**(*self*):

*self*.driver = webdriver.Firefox()

Import statements

The first 6 lines are simple import statements; their uses will become more apparent as we go along.

Class initialization

Notice the parameter on the class line (unittest.TestCase). This lets the program know that we are going to be running test cases. As a result the console will print out OK or Fail depending on the results of the cases.

Setup method

Using self as a parameter is going to be a constant for our methods. Within the method we set up a driver, this is going to be the web browser that is used to run our test case. For this example we are using Firefox because it is the easiest to use though Internet Explorer and Chrome are both options. When using Firefox make sure that you’re versions of Selenium and Firefox are compatible with one another.

Page 6

Section 1.3 (Running/Closing):

Example 1.3: code to run our tests as well as close them

def **etupwn**(*self*):

*self*.driver.close()

if \_\_name\_\_ == *“\_\_main\_\_”*:

unittest.main()

Unittest.main()

The if condition that comes before unittest.main()will always cause the code to be executed which is what we want. The code causes any method with a test\_ prefix to be tested.

Tear Down method

This code causes the web driver to close if the test passes or fails.

Page 7

Section 1.4 (Test Method):

Example 1.4: Shows the executable code

def **test\_login**(*self*):

driver = *self*.driver

driver.get(*“https://192.168.51.46”*)

wait = WebDriverWait(driver, 10)

*‘’’Username’’’*

element= wait.until(EC.presence\_of\_element\_located((By.NAME, *‘login’*)))

element.clear()

element.send\_keys(*‘admin’*)

*‘’’Password’’’*

element= wait.until(EC.presence\_of\_element\_located((By.NAME, *‘password’*)))

element.clear()

element.send\_keys(*‘raritan0’*)

*‘’’Click Button’’’*

element= wait.until(EC.element\_to\_be\_clickable((By.NAME, *‘action\_login’*)))

element.send\_keys(Keys.RETURN)

Test Login

Remember that if we want the code to be executed we need to put test\_ before the name of the method. Also note that the name of the test method does not have to be related to the class method.

Driver and Driver get

driver = *self*.driver takes the driver that we created in the setup method and creates an object of it.

Driver.get(*“https://192.168.51.46”*) opens our driver window and gives it a web address to go to.

Web Driver Wait

wait = WebDriverWait(driver, 10) creates a wait object. The first parameter is our driver and the second parameter is the amount of time (in seconds) that will be spent waiting for an event to happen before the program returns an error

Page 8

Username/ Password

These sections of code are used to enter data into the username and passwords respectively. Username and password are both input boxes.

Element= wait.until(EC.presence\_of\_element\_located((By.NAME, *‘login’*))) does a couple of different things. First it says wait.until(EC. EC stands for Expected Condition meaning that the program will wait (for the given time) for the condition to be true. If the time elapses before the condition is met then the test will return an error and fail. Presence\_of\_element\_located((By.NAME, *‘login’*)) is the EC that we are looking for. We are trying to find an element on the page. The element can be located in a number of different ways (Name, XPATH, Class, or ID) so use your discretion of which way is best. In this case we are finding the element by the name and the name is login. Once found the input box object is assigned to element and we can interact with the input box through element.

Element.clear() clears the input box of any other data that may already be present

element.send\_keys(*‘admin’*) sends the string into the input box. Make sure that it is set up as a string

Click Button

This section of code is used to click the login button.

Element= wait.until(EC.element\_to\_be\_clickable((By.NAME, *‘action\_login’*))) is very similar to the code we used before to find the input boxes. However, in this case our EC is to wait for the element to be clickable. This condition says to wait for the element to be present and for it to be clickable. It is an important distinction because we need the button to be present and clickable and these two things do not always happen at the same time.

Element.send\_keys(Keys.RETURN) sends the Enter key from the keyboard. Sending RETURN is very important because using the click command is not always as reliable.

Page 9

Section 1.5 (Navigating Menus):

Example 1.5: logs into the KX3 (explained in Section 1.4) and then shows how to access menus

import unittest

from selenium import webdriver

from selenium.webdriver.support.ui import WebDriverWait

from selenium.webdriver.common.by import By

from selenium.webdriver.support import expected\_conditions as EC

from selenium.webdriver.common.keys import Keys

class **ScratchNavigate**(unittest.TestCase):

def **etup**(*self*):

*self*.driver = webdriver.Firefox()

def **test\_Navigate**(*self*):

driver = *self*.driver

driver.get(*“https://192.168.51.46”*)

wait = WebDriverWait(driver, 10)

element= wait.until(EC.presence\_of\_element\_located((By.NAME, *‘login’*)))

element.clear()

element.send\_keys(*‘admin’*)

element= wait.until(EC.presence\_of\_element\_located((By.NAME, *‘password’*)))

element.clear()

element.send\_keys(*‘raritan0’*)

element= wait.until(EC.element\_to\_be\_clickable((By.NAME, *‘action\_login’*)))

element.send\_keys(Keys.RETURN)

*‘’’UM Menu’’’*

wait.until(EC.frame\_to\_be\_available\_and\_switch\_to\_it(*‘topnav’*))

element=wait.until(EC.presence\_of\_element\_located((By.LINK\_TEXT, *‘User Management’*)))

element.send\_keys(Keys.RETURN)

driver.switch\_to\_default\_content()

*‘’’Add New User’’’*

wait.until(EC.frame\_to\_be\_available\_and\_switch\_to\_it(*‘container’*))

element=wait.until(EC.presence\_of\_element\_located((By.LINK\_TEXT, *‘Add New User’*)))

element.send\_keys(Keys.RETURN)

driver.switch\_to\_default\_content()

def **teardown**(*self*):

*self*.driver.close()

if \_\_name\_\_ == *“\_\_main\_\_”*:

unittest.main()

Page 10

Notes:

There are a couple of different frames within the KX3 pages and so it is important to know which frames objects are located in. wait.until(EC.frame\_to\_be\_available\_and\_switch\_to\_it(*‘topnav’*)) switches from the default frame to the topnav frame once that frame becomes accessible.

Element=wait.until(EC.presence\_of\_element\_located((By.LINK\_TEXT, *‘User Management’*))) Searches for the Link\_Text (visible text) and assigns the object to the element variable.

Element.send\_keys(Keys.RETURN) sends the Enter key from the keyboard to the element object.

Driver.switch\_to\_default\_content() switches to the default frame, it is important to always do this when changing a page otherwise it will stay on the frame from before.

Page 11

Section 1.6 (Interacting with Checkboxes)

Example 1.6:

element= wait.until(EC.element\_to\_be\_clickable((By.XPATH, path)))

element.click()

This code is the best way to interact with checkboxes. You don’t have to find the element by XPATH. However, using element.send\_keys(Keys.RETURN) does not work on checkboxes so you need to use element.click().

Page 12

Section 1.7 (Interacting with Select boxes)

There are a couple of different things to do with select boxes. They are all outlined in the following examples. Be sure to include this import statement: from selenium.webdriver.support.ui import Select

Example 1.7:

This code returns all of the options within the select box.

From selenium.webdriver.support.ui import Select

wait.until(EC.element\_to\_be\_clickable((By.ID, *‘interfaceOption’*)))

element= Select(driver.find\_element(By.ID, *‘interfaceOption’*))

return element.options

wait.until(EC.element\_to\_be\_clickable((By.ID, *‘interfaceOption’*))) waits for the element to be clickable.

Element= Select(driver.find\_element(By.ID, *‘interfaceOption’*)) assigns the object to element. Using Select( identifies the element as a select box.

Return element.options returns all of the options in the select box as an array/list.

Example 1.8: finds the select box and returns its current value

element= wait.until(EC.element\_to\_be\_clickable((By.ID, *‘interfaceOption’*)))

return element.get\_attribute(*‘value’*)

Example 1.9: finds the select box and selects a given option

wait.until(EC.element\_to\_be\_clickable((By.ID, *‘interfaceOption’*)))

element=Select(driver.find\_element(By.ID, *‘interfaceOption’*))

element.select\_by\_visible\_text(text)

element.select\_by\_visible\_text(text) searches for the visible text option in the select box and chooses it. The value text is not always the same as the value attribute from Example 1.8

Page 13

Section II:

Utility Methods

Section 2.1 (Summary):

The utility methods were developed in order to make the code simpler. They take pieces of commonly used code and put them in easy to access methods. The first sections are used to document these methods and the last one serves as an example on how to use them.

Section 2.2 (ClickOn):

The ClickOn methods are used to click on objects using their object name, class, XPATH, etc.

Format:

def **Click\_Identifier**(*self*, driver, wait, text):

try:

element= wait.until(EC.element\_to\_be\_clickable((By.Identifier, text)))

element.send\_keys(Keys.RETURN)

except:

Click\_Identifier(*self*, driver, wait, text)

Methods:

Click\_Link\_Text(self, driver, wait, text)

Click\_XPATH(self, driver, wait, text)

Hard\_Click\_XPATH(self, driver, wait, text)

Click\_Name(self, driver, wait, text)

Click\_ID(self, driver, wait, text)

Notes:

The text parameter is the identifier to found the desired element whether it be the Class, Name, XPATH, etc.

There are two methods which use XPATH as the identifier. Click\_XPATH uses element.click() and Hard\_Click\_XPATH uses element.send\_keys(Keys.RETURN). Checkboxes should use Click\_XPATH since Keys.RETURN does not work on them.

Page 14

Section 2.3 (Driver Control):

The Driver Control is used to simplify the initialization of the driver object. It create a web driver using the Firefox browser and use <http://IP> as the web address

Code:

from selenium import webdriver

def **setUpFireFox**(*self*, IP):

*self*.driver = webdriver.Firefox()

driver = *self*.driver

driver.get(*"http://"*+IP)

return driver

Notes:

The IP parameter is the IP address of your KX3 device (should be formatted as a string).

Page 15

Section 2.4 (File Read):

Since each KX3 device is going to have its own IP, Username, and Password I created a file which makes it easy to format your personal settings.

File Format:

<IP Address>192.168.51.46</IP Address>

<adminName>admin</adminName>

<adminPwd>raritan0</adminPwd>

Code:

f=open(*'C:/Users/brendanm/Documents/Eclipse/KX3/src/Utilities/Information.txt'*, *'r'*)

line1=f.readline()

begin=line1.find(*'>'*)

end=line1.find(*'</'*)

IP=line1[begin+1:end]

line2=f.readline()

begin=line2.find(*'>'*)

end=line2.find(*'</'*)

adminUser=line2[begin+1:end]

line3=f.readline()

begin=line3.find(*'>'*)

end=line3.find(*'</'*)

adminPwd=line3[begin+1:end]

Notes:

The file is meant to be easily readable and the information is parsed.

For more information on interacting with files see Section IV.

Section 2.5 (File Setup):

Will send the results of test cases to an output file

Code:

import time

date= time.strftime(*"%m-%d-%Y"*)

time= time.strftime(*"%I:%M:%S %p"*)

f= open(*'C:/Users/brendanm/Documents/TestCases/KX3-%s.txt'* %date,*'w'*)

Notes:

Creates a writable file which is named by the date, specific outputs are created within the tests themselves.

For more information on using files see Section IV.

Page 16

Section 2.6 (Input):

Series of methods used to interact with Input Boxes

Format:

def **Input\_By\_Type**(*self*, driver, wait, identifier, text):

try:

element= wait.until(EC.presence\_of\_element\_located((By.Type, identifier)))

element.clear()

element.send\_keys(text)

except:

Input\_By\_Name(*self*, driver, wait, identifier, text)

Methods:

Input\_By\_Name(self, driver, wait, identifier, text)

Input\_By\_ID(self, driver, wait, identifier, text)

Input\_By\_XPATH(self, driver, wait, identifier, text)

Notes:

There are two parameter for these methods; identifier and text. Identifier is used to locate the Input box whether it is by the Name, ID, or XPATH. Text is the string which is being sent to the input box.

Page 17

Section 2.7 (Login):

Contains two methods, one for logging into the KX3 and one for logging out

Code:

from selenium.webdriver.support import expected\_conditions as EC

from Utilities import ClickOn

from Utilities import Input

from Utilities import FileRead

IP= FileRead.IP+*'/admin'*

IP2=FileRead.IP

PWD=FileRead.adminPwd

UNAME=FileRead.adminUser

def **login**(*self*, driver, wait, uname=UNAME, password=PWD):

Input.Input\_By\_Name(*self*, driver, wait, *'login'*, uname)

Input.Input\_By\_Name(*self*, driver, wait, *'password'*, password)

ClickOn.Click\_Name(*self*, driver, wait, *'action\_login'*)

driver.switch\_to.default\_content()

def **logout** (*self*, driver, wait):

try:

driver.switch\_to.default\_content()

wait.until(EC.frame\_to\_be\_available\_and\_switch\_to\_it(*'container'*))

ClickOn.Click\_Link\_Text(*self*, driver, wait, *'Logout'*)

except:

logout(*self*, driver, wait)

Notes:

The login method has uname, and password as parameters. Unless otherwise specified these variables will default to the values provided by the Information.txt file.

IP and IP2 are called by the test cases using Login.IP or Login.IP2. Login.IP is more commonly used since adding the /admin to the address disables many of the java popups.

Page 18

Section 2.8 (Navigate):

This class contains a single method which is used to navigate through the various menus of the KX3 device.

Code:

from selenium.webdriver.support import expected\_conditions as EC

from selenium.webdriver.common.by import By

from Utilities import ClickOn

def **Navigate**(*self*, driver, wait, path):

driver.switch\_to.default\_content()

wait.until(EC.frame\_to\_be\_available\_and\_switch\_to\_it(*'topnav'*))

separator=path.find(*';'*)

menu=path[0:separator]

wait.until(EC.presence\_of\_element\_located((By.LINK\_TEXT, menu)))

assert menu in driver.page\_source

ClickOn.Click\_Link\_Text(*self*, driver, wait, menu)

driver.switch\_to.default\_content()

path=path[separator+1:]

while len(path)!=0:

wait.until(EC.frame\_to\_be\_available\_and\_switch\_to\_it(*'container'*))

separator=path.find(*';'*)

menu=path[0:separator]

wait.until(EC.presence\_of\_element\_located((By.LINK\_TEXT, menu)))

assert menu in driver.page\_source

ClickOn.Click\_Link\_Text(*self*, driver, wait, menu)

driver.switch\_to.default\_content()

path=path[separator+1:]

Notes:

The path parameter is used to specify which menus are going to be accessed. The path looks for each menu name to be separated by a semicolon. ’Device Settings;Connection Scripts;’ would search for Device settings in the top nav bar and then click on the connection scripts link.

A semicolon is needed at the end of the path

Though two is the most common number of menus being accessed this method will work with one and three menus in the path as well.

Page 19

Section 2.9 (Strong Passwords):

This utility is used to help create passwords. This utility is usually not needed but is helpful when used.

Code:

import random

def **Generate**():

password=*'Ab\*'*

for x in range(0,8):

number= random.randint(1,9)

password= password+str(number)

return password

def **Long**():

password=*''*

for x in range(0,100):

password=password+*'a'*

return password

Methods:

Generate makes a random password which the KX3 will classify as a strong password

Long makes a password comprised of 100 a’s

Page 20

Section 2.10 (User Management)

These methods are used to add and delete users/ user groups.

Code:

from selenium.webdriver.support import expected\_conditions as EC

from selenium.webdriver.common.by import By

from selenium.webdriver.support.ui import Select

from selenium.webdriver.common.alert import Alert

from Utilities import ClickOn

from Utilities import Navigate

from Utilities import Input

def **new\_user\_group**(*self*, driver, wait, permission=*'FV\_2\_acl#usr\_mngmnt'*, name=*'groupName'*):

try:

Navigate.Navigate(*self*, driver, wait, *'User Management;Add New User Group;'*)

wait.until(EC.frame\_to\_be\_available\_and\_switch\_to\_it(*'container'*))

Input.Input\_By\_Name(*self*, driver, wait, *'FV\_2\_umgroups'*, name)

driver.find\_element(By.ID, *'g\_p\_acl\_cpt\_bar'*).click()

driver.find\_element(By.NAME, permission).click()

driver.find\_element(By.NAME, *'action\_create\_group'*).click()

driver.switch\_to.default\_content()

except:

new\_user\_group(*self*, driver, wait, permission, name)

def **new\_user**(*self*,driver, wait, uname=*'bmccann'*, pwd=*'abc123'*, group=*'groupName'*):

try:

Navigate.Navigate(*self*, driver, wait, *'User Management;Add New User;'*)

wait.until(EC.frame\_to\_be\_available\_and\_switch\_to\_it(*'container'*))

Input.Input\_By\_Name(*self*, driver, wait, *'FV\_1\_umusers'*, uname)

Input.Input\_By\_Name(*self*, driver, wait, *'FV\_2\_umusers'*, *'Brendan McCann'*)

Input.Input\_By\_Name(*self*, driver, wait, *'FV\_3\_umusers'*, pwd)

Input.Input\_By\_Name(*self*, driver, wait, *'FV\_4\_umusers'*, pwd)

wait.until(EC.presence\_of\_element\_located((By.NAME, *'FV\_7\_umusers'*)))

element=Select(driver.find\_element(By.NAME, *'FV\_7\_umusers'*))

element.select\_by\_visible\_text(group)

driver.find\_element(By.NAME, *'action\_create\_user'*).click()

driver.switch\_to.default\_content()

except:

new\_user(*self*, driver, wait, uname, pwd, group)

def **delete\_user**(*self*, driver, wait):

Navigate.Navigate(*self*, driver, wait, *'User Management;User List;'*)

wait.until(EC.frame\_to\_be\_available\_and\_switch\_to\_it(*'container'*))

Page 21

ClickOn.Click\_XPATH(*self*, driver, wait, *"//table[@class='dtbl']/tbody/tr[3]/td[2]/input[@class='input\_checkbox']"*)

ClickOn.Click\_Name(*self*, driver, wait, *'delete'*)

wait.until(EC.alert\_is\_present())

Alert(driver).accept()

driver.switch\_to.default\_content()

try:

wait.until(EC.frame\_to\_be\_available\_and\_switch\_to\_it(*'container'*))

assert *'deleted'* in driver.page\_source

except:

delete\_user(*self*, driver, wait)

driver.switch\_to.default\_content()

def **delete\_user\_group**(*self*, driver, wait):

Navigate.Navigate(*self*, driver, wait, *'User Management;User Group List;'*)

wait.until(EC.frame\_to\_be\_available\_and\_switch\_to\_it(*'container'*))

ClickOn.Click\_XPATH(*self*, driver, wait, *"//table[@class='dtbl']/tbody/tr[4]/td[2]/input[@class='input\_checkbox']"*)

ClickOn.Click\_Name(*self*, driver, wait, *'delete'*)

wait.until(EC.alert\_is\_present())

Alert(driver).accept()

driver.switch\_to.default\_content()

try:

wait.until(EC.frame\_to\_be\_available\_and\_switch\_to\_it(*'container'*))

assert *'deleted'* in driver.page\_source

except:

delete\_user\_group(*self*, driver, wait)

driver.switch\_to.default\_content()

Methods:

New\_user\_group(self, driver, wait, permission, name) creates a new user group

Permission clicks the checkbox related to group permissions. The search is done by Name and defaults as 'FV\_2\_acl#usr\_mngmnt'. Name is a parameter which serves as the group name.

New\_user(self, driver, wait, uname, password, group) creates a new user

Uname is a parameter which gives the user a name defaults to 'bmccann'. Password is a parameter which gives the user a password and defaults to 'abc123'. Group is a parameter which assigns the user to a user group and defaults to 'groupName'.

Delete\_user(self, driver, wait) deletes the first user in the list (excluding admin)

Delete\_user\_group(self, driver, wait) deletes the first user group in the list (excluding Unknown and Admin)

Page 22

Section 2.11 (Examples):

EX 2.1: Gives an example for using ClickOn, Login, Navigate, and DriverControl

import unittest

from selenium.webdriver.support.ui import WebDriverWait

from selenium.webdriver.common.by import By

from selenium.webdriver.support import expected\_conditions as EC

from selenium.webdriver.support.ui import Select

from Utilities import ClickOn

from Utilities import Login

from Utilities import Navigate

from Utilities import DriverControl

class **TestSpace**(unittest.TestCase):

def **test\_main**(*self*):

driver=DriverControl.setUpFireFox(*self*, Login.IP)

text=[*‘Ip:’*, *‘Ipv6’*, *‘Kernel Interface table’*, *‘connections’*]

options=[*‘statistics’*, *‘route’*, *‘interfaces’*,*’ports’*]

wait = WebDriverWait(driver, 5)

Login.login(*self*, driver, wait)

Navigate.Navigate(*self*, driver, wait, *‘Diagnostics;Network\_Statistics;’*)

wait.until(EC.frame\_to\_be\_available\_and\_switch\_to\_it(*‘container’*))

select = Select(driver.find\_element(By.ID, *‘options’*))

assert select.first\_selected\_option.get\_attribute(*“value”*)==*’statistics’*

for x in range(0, len(text)):

ClickOn.Click\_Name(*self*, driver, wait, *‘action\_resfresh’*)

wait.until(EC.presence\_of\_element\_located((By.XPATH, *“//table[@class=’tmpl\_tab’]/tbody/tr[5]/td/div/br”*)))

assert text[x] in driver.page\_source

if x!=3:

select.select\_by\_value(options[x+1])

assert text[x] in driver.page\_source

Login.logout(*self*, driver, wait)

def **eardown**(*self*):

*self*.driver.close()

if \_\_name\_\_ == *“\_\_main\_\_”*:

unittest.main()

Page 23

EX 2.2: An example which highlights the user management features

import unittest

from selenium.webdriver.support.ui import WebDriverWait

from selenium.webdriver.support import expected\_conditions as EC

from Utilities import Login, UserManagement as UM, DriverControl as DC

class **Test2668**(unittest.TestCase):

def **test\_main**(*self*):

file= FileSetup.f

driver=DC.setUpFireFox(*self*, Login.IP)

wait = WebDriverWait(driver, 5)

Login.login(*self*, driver, wait)

UM.new\_user(*self*, driver, wait, *'bmccann'*, *'abc123'*, *'<Unknown> (default setting)'*)

UM.new\_user(*self*, driver, wait, *'bmccann'*, *'abc123'*, *'<Unknown> (default setting)'*)

wait.until(EC.frame\_to\_be\_available\_and\_switch\_to\_it(*'container'*))

assert *'Error:'* in driver.page\_source

*'''Asserts that a duplicate user causes an error'''*

UM.delete\_user(*self*, driver, wait)

Login.logout(*self*, driver, wait)

def **tearDown**(*self*):

*self*.driver.close()

if \_\_name\_\_ == *"\_\_main\_\_"*:

unittest.main()

EX 2.3: Variation of the login method. It highlights the ability to use the Input method

def **login**(*self*, driver, wait, uname=UNAME, password=PWD):

Input.Input\_By\_Name(*self*, driver, wait, *'login'*, *'admin'*)

Input.Input\_By\_Name(*self*, driver, wait, *'password'*, *'raritan0'*)

ClickOn.Click\_Name(*self*, driver, wait, *'action\_login'*)

driver.switch\_to.default\_content()

Page 24

Section III:

Page Objects

Section 3.1 (summary):

This section is made to list out the different types of page object methods used as well as their implementation. There is a method for every element on every page. Most of the methods implement utility methods.

Section 3.2 (naming convention):

Page object methods have a general naming method which is used to make finding the objects easier

Step 1: Action being taken click, set, get, etc.

Step 2: Object that the action is being applied to Btn (button), Chckbx (check box), Txtbx (text box), etc.

Step 3: Underscore identifier/ label of the element \_PWD, \_Usrname, \_Delete

Examples:

clickCheckbx\_Apply

clickBtn\_OK

getTxtbx\_SSHPort

Section 3.3 (Buttons):

Buttons are probably the easiest to work with especially when using the ClickOn methods.

def **clickBtn\_Add**(*self*, driver, wait):

ClickOn.Click\_Name(*self*, driver, wait, *'Whatever the name is'*)

Page 25

Section 3.4 (Checkboxes):

Checkboxes have two methods; get and click. Get will return whether or not the box is currently selected and click will click on the box

def **getCheckbx\_EnblSNMPV3**(*self*, driver, wait):

element= wait.until(EC.element\_to\_be\_clickable((By.NAME, *'FV\_8\_snmpagent'*)))

return element.is\_selected()

def **clickCheckbx\_EnblSNMPV3**(*self*, driver, wait):

element= wait.until(EC.element\_to\_be\_clickable((By.NAME, *'FV\_8\_snmpagent'*)))

element.click()

It is also very common to see checkboxes within tables, in which case they will require an additional parameter row. Row is an integer that corresponds to the row in the table. Try to be aware of any offsets in the row count and adjust accordingly. For example the first row of the table may be labeled row 0 instead of row 1.

def **getCheckbx\_Apply**(*self*, driver, wait, row: int):

element= wait.until(EC.element\_to\_be\_clickable((By.ID, *'apply\_to\_%s'* %row)))

return element.is\_selected()

def **clickCheckbx\_Apply**(*self*, driver, wait, row: int):

element= wait.until(EC.element\_to\_be\_clickable((By.ID, *'apply\_to\_%s'* %row)))

element.click()

Section 3.5 (Textboxes):

Textboxes have two methods; get and set. Get will return the current text located in the textbox and set will send a string to the textbox via the Input method.

def **getTxtbx\_PrivPassPhrase**(*self*, driver, wait):

element=wait.until(EC.presence\_of\_element\_located((By.NAME, *'FV\_14\_snmpagent'*)))

return element.get\_attribute(*'value'*)

def **setTxtbx\_PrivPassPhrase**(*self*, driver, wait, text:str):

Input.Input\_By\_Name(*self*, driver, wait, *'FV\_14\_snmpagent'*, text)

Notes:

The text parameter in the setTxtbx method is the String being sent to the text box

Page 26

Section 3.6 (Select boxes):

Select boxes have three methods and their naming convention is a little different. They are named getCrrntSelbx, getAllSelbx, and setSelbx. getCrrntSelbx methods return the currently selected option. getAllSelbx returns all of the options within the select box in the form of an array/ list. setSelbx will change the selected option within the select box.

Code:

def **getAllSelbx\_PrivProtocol**(*self*, driver, wait):

wait.until(EC.element\_to\_be\_clickable((By.NAME, *'FV\_12\_snmpagent'*)))

element= Select(driver.find\_element(By.NAME, *'FV\_12\_snmpagent'*))

return element.options

def **getCrrntSelbx\_PrivProtocol**(*self*, driver, wait):

element= wait.until(EC.element\_to\_be\_clickable((By.NAME, *'FV\_12\_snmpagent'*)))

return element.get\_attribute(*'value'*)

def **setSelbx\_PrivProtocol**(*self*, driver, wait, text: str):

wait.until(EC.element\_to\_be\_clickable((By.NAME, *'FV\_12\_snmpagent'*)))

element=Select(driver.find\_element(By.NAME, *'FV\_12\_snmpagent'*))

element.select\_by\_visible\_text(text)

Notes:

The get methods will return the options based on their values while the set method chooses an option based on its visible text. Values and visible text do not always match up with one another.

Section 3.7 (Links):

These links are different than the ones used to access menus. They are very common within the user management pages and implement click on methods.

Code:

def **clickLink\_UsrName**(*self*, driver, wait, row: int):

row= row+1

ClickOn.Hard\_Click\_XPATH(*self*, driver, wait, *"//table[@class='dtbl']/tbody/tr[%s]/td[3]/a"* %row)

Notes:

Links are commonly found in tables and so they have an extra row parameter. Many of the tables have buffer rows so it’s important to adjust the integer accordingly.

Page 27

Section 3.8 (Data):

Data is words and values stored within the page. They are commonly found in tables and cannot be directly changed but can be retrieved. As a result data only has a get method.

Code:

def **getData\_ScriptsInUse**(*self*, driver, wait, row: int):

row= row+1

element= wait.until(EC.presence\_of\_element\_located((By.XPATH, *"//table[@class='dtbl']/tbody/tr[%s]/td[5]"* %row)))

return element.text

Notes:

Data is commonly found in tables and so they have an extra row parameter. Many of the tables have buffer rows so it’s important to adjust the integer accordingly.

Data is retrieved by finding the location of the element and then returning the text of that element by the return element.text command.

Section 3.9 (Sort):

Most tables in the KX3 system can be sorted by category headers. Sort type methods click on these headers by using the ClickOn methods.

Code:

def **clickSort\_SortByNumber**(*self*, driver, wait):

ClickOn.Click\_Link\_Text(*self*, driver, wait, *'No.'*)

Notes:

Clicking a sort link the first time will cause values to be ordered A-Z or 1-9 based on the category but hitting the link a second time will reverse the order.

Page 28

Section 3.10 (Menus):

Menu methods are used to access all of the various menus and submenus of the KX3. These methods are made easy by using the Navigate method.

Code:

def **DeviceSettings**(*self*, driver, wait):

Navigate.Navigate(*self*, driver, wait, *'Device Settings;'*)

def **ConnectionScripts**(*self*, driver, wait):

Navigate.Navigate(*self*, driver, wait, *'Device Settings;Connection Scripts;'*)

Notes:

Menu methods are named based on the page that you are accessing.

Section 3.11 (Examples):

EX 3.1: Attempts to change the time zone settings

import unittest

from selenium.webdriver.support.ui import WebDriverWait

from selenium.webdriver.support import expected\_conditions as EC

from SeleniumPython.KVM.KX3.Utilities import Login, DriverControl as DC

from SeleniumPython.KVM.KX3.PageObjects.DeviceSettings import DeviceSettingsMenu, DateTime

class **Test2594**(unittest.TestCase):

def **test\_main**(*self*):

driver=DC.setUpFireFox(*self*, Login.IP)

wait = WebDriverWait(driver, 10)

Login.login(*self*, driver, wait)

DeviceSettingsMenu.DateTime(*self*, driver, wait)

wait.until(EC.frame\_to\_be\_available\_and\_switch\_to\_it(*'container'*))

assert *'Time Zone'* in driver.page\_source

DateTime.setSelbx\_TimeZone(*self*, driver, wait, *'Pacific/Midway'*)

DateTime.clickBtn\_Cancel(*self*, driver, wait)

driver.switch\_to.default\_content()

*'''Selects a time zone and cancels'''*

DeviceSettingsMenu.DateTime(*self*, driver, wait)

wait.until(EC.frame\_to\_be\_available\_and\_switch\_to\_it(*'container'*))

DateTime.setSelbx\_TimeZone(*self*, driver, wait, *'Pacific/Midway'*)

DateTime.clickBtn\_OK(*self*, driver, wait)

driver.switch\_to.default\_content()

Page 29

wait.until(EC.frame\_to\_be\_available\_and\_switch\_to\_it(*'container'*))

assert *'Operation completed successfully'* in driver.page\_source

driver.switch\_to.default\_content()

*'''Asserts a successful change'''*

wait.until(EC.frame\_to\_be\_available\_and\_switch\_to\_it(*'container'*))

DateTime.setSelbx\_TimeZone(*self*, driver, wait, *'America/New\_York'*)

DateTime.clickBtn\_OK(*self*, driver, wait)

driver.switch\_to.default\_content()

wait.until(EC.frame\_to\_be\_available\_and\_switch\_to\_it(*'container'*))

assert *'Operation completed successfully'* in driver.page\_source

*'''Need to change it back'''*

Login.logout(*self*, driver, wait)

def **tearDown**(*self*):

*self*.driver.close()

if \_\_name\_\_ == *"\_\_main\_\_"*:

unittest.main()

Notes:

Page objects do not handle the switching of frames so coders need to be aware of the current and desired frames.

Take note of how the Page Objects are imported:

from SeleniumPython.KVM.KX3.PageObjects.DeviceSettings import DeviceSettingsMenu, DateTime

From should be the path leading to the parent menu. Import takes DateTime and DeviceSettingsMenu. DateTime contains the methods for all of the elements on the Date/ Time page. DeviceSettingsMenu has all of the navigate methods for the Device Settings Menu and submenus.

Methods need to be preceded by the page name that you imported: DateTime.setSelbx\_TimeZone(*self*, driver, wait, *'America/New\_York'*)

As a result, different pages can have the same method name because they will be distinguished by whichever page you are on. However, the same page cannot have two methods of the same name.

Page 30

Section IV:

File Usage

Section 4.1 (Summary):

These tests use files in two different ways. They retrieve data such as username, password, and IP from an information file. Test results are also sent to a separate file. These things are done using FileRead and FileSetup utilities.

Section 4.2 (FileRead):

FileRead uses Information.txt to read in unique information such as IP, username, and password.

Information.txt:

<IP Address>192.168.51.46</IP Address>

<adminName>admin</adminName>

<adminPwd>raritan0</adminPwd>

FileRead.py:

f=open(*'C:/Users/brendanm/Documents/Eclipse/SVNStructure/src/SeleniumPython/KVM/KX3/Utilities/Information.txt'*, *'r'*)

line1=f.readline()

begin=line1.find(*'>'*)

end=line1.find(*'</'*)

IP=line1[begin+1:end]

line2=f.readline()

begin=line2.find(*'>'*)

end=line2.find(*'</'*)

adminUser=line2[begin+1:end]

line3=f.readline()

begin=line3.find(*'>'*)

end=line3.find(*'</'*)

adminPwd=line3[begin+1:end]

Notes:

The information file needs to be modified so that it reflects the properties of your KX3 file. In addition the f=open( needs to reflect your path.

The ‘r’ parameter in the open command stands for read; it means that the file can only be read and not edited.

Page 31

Section 4.3 (FileSetup):

FileSetup creates a file and writes out the test results of each test case

FileSetup.py:

import time

date= time.strftime(*"%m-%d-%Y"*)

time= time.strftime(*"%I:%M:%S %p"*)

f= open(*'C:/Users/brendanm/Documents/TestCases/KX3-%s.txt'* %date,*'w'*)

EX 4.1:

import sys

import unittest

from selenium.webdriver.support.ui import WebDriverWait

from selenium.webdriver.common.by import By

from selenium.webdriver.support import expected\_conditions as EC

from Utilities import Login, FileSetup, DriverControl as DC

from PageObjects.Diagnostics import DiagnosticsMenu, NetworkStatistics

class **Test2685**(unittest.TestCase):

def **test\_main**(*self*):

file= FileSetup.f

text=[*'Ip:'*, *'IPv6'*, *'Kernel Interface table'*, *'connections'*]

options=[*'--statistics'*, *'--route'*, *'--interfaces'*,*'--ports'*]

driver=DC.setUpFireFox(*self*, Login.IP)

wait = WebDriverWait(driver, 5)

try:

Login.login(*self*, driver, wait)

DiagnosticsMenu.NetworkStatistics(*self*, driver, wait)

wait.until(EC.frame\_to\_be\_available\_and\_switch\_to\_it(*'container'*))

assert NetworkStatistics.getCrrntSelbx\_Option(*self*, driver, wait)==*'statistics'*

for x in range(0, len(text)):

NetworkStatistics.clickBtn\_Refresh(*self*, driver, wait)

wait.until(EC.presence\_of\_element\_located((By.XPATH, *"//table[@class='tmpl\_tab']/tbody/tr[5]/td/div/br"*)))

assert text[x] in driver.page\_source

if x!=3:

NetworkStatistics.setSelbx\_Option(*self*, driver, wait, options[x+1])

assert text[x] in driver.page\_source

*'''Potential to be changed'''*

Login.logout(*self*, driver, wait)

file.write(*'DFTC003 Network Statistics, Passed, ('*+FileSetup.date+*' '*+FileSetup.time+*')\n'*)

Page 32

except:

file.write(*'DFTC003 Network Statistics, Failed, '*+str(sys.exc\_info()[0])+*', ('*+FileSetup.date+*' '*+FileSetup.time+*')\n'*)

def **tearDown**(*self*):

*self*.driver.close()

if \_\_name\_\_ == *"\_\_main\_\_"*:

unittest.main()

Output Samples:

Pass:

DFTC003 Network Statistics, Passed, (07-07-2016 01:21:58 PM)

Fail:

DFTC003 Network Statistics, Failed, <class 'AssertionError'>, (07-07-2016 01:23:45 PM)

Notes:

file= FileSetup.f takes the file from FileSetup and assigns it to a local variable called file.

The file created is named based off of the date. Make sure you change the path in FileSetup.py.

Most of the TestCase is placed within a try except case. The idea is that if the test succeeds then it will execute: file.write(*'DFTC003 Network Statistics, Passed, ('*+FileSetup.date+*' '*+FileSetup.time+*')\n'*). This code writes the name, passed, date, and time of the test. If an error occurs at any point then the except clause is called and will execute: file.write(*'DFTC003 Network Statistics, Failed, '*+str(sys.exc\_info()[0])+*', ('*+FileSetup.date+*' '*+FileSetup.time+*')\n'*). This line writes name, failed, type of error encountered, date, and time of the test.

Page 33

Section V:

Batch Testing

Section 5.1 (Summary):

Batch testing enables multiple test cases to be run together. Batch testing requires two different modules in order to be run; BatchImport and BatchTest.

Section 5.2 (BatchImport):

BatchImport is used to help setup the tests for the BatchTest module. It imports the tests that we want to be running and then places them in an array.

Code:

from TestCases.BianorAutomatedTestCases.Network.Diagnostics.NetworkStatistics.KX3\_2405\_Traceroute\_S import Test2405

from TestCases.RaritanAutomatedTestCases.Diagnositcs.NetworkStatistics.KX3\_2685\_NetworkStatistics\_S import Test2685

imports=[Test2405.Test2405, Test2685.Test2685]

Notes:

Test2405.Test2405 appears redundant but takes on a naming structure of Class.MainMethod

Page 34

Section 5.3 (BatchTest)

The code that is used to create and execute a test suite

Code:

import unittest

from BatchTesting import BatchImports

class **Test\_Suite**(unittest.TestCase):

def **test\_main**(*self*):

*self*.suite = unittest.TestSuite()

for x in BatchImports.imports:

*self*.suite.addTest(unittest.defaultTestLoader.loadTestsFromTestCase(x))

runner = unittest.TextTestRunner()

runner.run (*self*.suite)

if \_\_name\_\_ == *"\_\_main\_\_"*:

unittest.main()

Notes:

BatchTesting is the name of the package that BatchImports and BatchTest are located in. You can rename this package as long as both modules remain in it.

Loops in Python work differently than in Java. The loop:

for x in BatchImports.imports has x iterating through all of the values present in the imports array. As a result knowing the length of the array is not necessary.

Add whichever test cases you want to the imports array and they will be run

Each test opens and closes its own window

Tests run in the order in which they are added

All tests run even if previous ones fail

The results of all of the tests will appear in a single file

Page 35

Section VI:

SVN Instructions

Section 6.1 (Summary):

This section contains some helpful information for using SVN. These sections take on a more step-by-step type of instruction. In addition Section 6.2 explains how to install the SVN Exploring onto Eclipse.

Section 6.2 (Installing SVN):

1. Help-> Install New Software
2. In the Work with bar enter the following link: <http://subclipse.tigris.org/update_1.6.x>
3. Select the Core SVN Kit and the Subclipse checkboxes
4. Hit next, agree to the license, hit Finish
5. Once the download is complete look in the top right hand corner of Eclipse and click Open Perspective, the SVN Repository Exploring Option should now be available

Look at this link for additional support: <https://javatutorial.net/install-svn-eclipse>

Section 6.3 (Accessing Repositories):

1. In the left hand side-menu click on the SVN Repositories tab
2. Right click the underlying white space, under new select repository location
3. Enter the URL of the repository and then hit finish
4. To checkout right click the repository icon and click checkout
5. Click the checkout as a project in the workspace option and give the project a name
6. You should now see the project in your workspace, this project is connected to the project on SVN and the two can commit, update, and synchronize with one another

Page 36

Section 6.4 (Synchronizing with Repository):

1. Choose a file from your workspace to work with and open it
2. Right click the file, Team-> Synchronize with Repository, this will open a comparison between your file and the Repository file
3. In the top right hand corner of the comparison window you should see buttons navigate through the differences in the files.
   1. Next difference
   2. Previous difference
   3. Next change
   4. Previous change
4. To the left of these navigation buttons should be the buttons to make changes. Note that the file on the left is your copy and the one on the right is the repository file.
   1. Copy all from Left to Right
   2. Copy all non-conflicting changes from right to left
   3. Copy current change from left to right
   4. Copy current change from right to left
5. Once you have completed these steps you should have an up to date version of the file and can close the comparison window.

Section 6.5 (Changing the Repository):

1. Choose a file from your workspace to work with and open it
2. Make any edits you want to make to your version of the file and then save it
3. Right click the file/ folder, Team-> Commit
4. Add a comment to your commit, commits cannot be made without a comment
5. To see the changes reflected in the repository right click the Repository icon and select refresh. After that you should see the commit take effect.

Page 37

Section 6.6 (Viewing History):

1. If you are in your work space right click the file, Team-> Show History
   1. Note: selecting Show Local History will only reflect your changes
2. If you are in the repository space right click the file, click Show History
3. In the bottom part of the window a tab called history should open itself up. This tab will contain a list of all the changes, when they were made, who they were made by, and the comments
4. Double clicking any of the rows in the History grid will open that version of the file

Page 38

Appendix A:

Helpful Reference Links

Python Tutorial: <https://docs.python.org/3.5/tutorial/index.html>

Python Selenium Tutorial: <http://selenium-python.readthedocs.io/installation.html>

Using Files in Python: <https://docs.python.org/3/tutorial/inputoutput.html>

Batch Testing: <http://roadtoautomation.blogspot.com/2014/09/road-to-create-test-suite-for-python.html>

Appendix B:

Example List

Example 1.1: complete test case built from scratch.

Located: Section 1.1, Page 4

Example 1.2: code needed to setup class and driver.

Located: Section 1.2, Page 5

Example 1.3: code to run tests as well as close them.

Located: Section 1.3, Page 6

Example 1.4: actual tested code.

Located: Section 1.4, Page 7

Example 1.5: logs into the KX3 and then accesses menus

Located: Section 1.5, Page 9

Example 1.6: code to click on check boxes

Located: Section 1.6, Page 11

Example 1.7: code to return all of the options within a select box.

Located: Section 1.7, Page 12

Page 39

Example 1.8: returning current value of select box

Located: Section 1.7, Page 12

Example 1.9: chooses the given option from a select box

Located: Section 1.7, Page 12

Example 2.1: uses ClickOn, Login, Navigate, and DriverControl

Located: Section 2.11, Page 22

Example 2.2: Shows User Management methods

Located: Section 2.11, Page 23

Example 2.3: shows the Input method

Located: Section 2.11, Page 23

Example 3.1: Test Case to change time zone settings

Located: Section 3.11, Page 28

Page 40

Appendix C:

Coding Template

import sys

import unittest

from selenium.webdriver.support.ui import WebDriverWait

from selenium.webdriver.common.by import By

from selenium.webdriver.support import expected\_conditions as EC

from Utilities import Login, FileSetup, DriverControl as DC, any other utilities needed

from PageObjects.MenuName import MenuNameMenu, SubMenus

class **TestNumber**(unittest.TestCase):

def **test\_main**(*self*):

file= FileSetup.f

driver=DC.setUpFireFox(*self*, Login.IP)

wait = WebDriverWait(driver, 5)

try:

Testing Code

file.write(*'TestName,Passed,('*+FileSetup.date+*' '*+FileSetup.time+*')\n'*)

except:

file.write(*'TestName, Failed, '*+str(sys.exc\_info()[0])+*', ('*+FileSetup.date+*' '*+FileSetup.time+*')\n'*)

def **tearDown**(*self*):

*self*.driver.close()

if \_\_name\_\_ == *"\_\_main\_\_"*:

unittest.main()

Note:

Highlighted items are things that you will have to change for yourself. Everything else should be pretty constant.