PharmaSUG China 2022 - Paper AD-127

Automation for Browser Using Selenium Python – Download EDC Data in RAVE for Example

Zhouhao Pan, BeiGene

ABSTRACT

To support the clinical data clean, analysis and monitor, we must download the clinical data from EDC system to our SAS server frequently to make sure it is up to date. And the snapshot analysis also requires EDC data being downloaded at a specified time. It makes EDC data downloading can happen at any time and we must be on standby before getting the confirmation that EDC data is ready. It wastes a lot of time and resource when waiting the request to perform EDC data downloading.

What if we can make it automated and no more resource need to be on standby? Selenium is a powerful tool to control web browsers through programs and performing browser automation. It is functional for many browsers, works on all major OS.. Selenium Python bindings provide a convenient API to access Selenium WebDrivers like Firefox, IE, Chrome, Remote and etc. Through Selenium Python API we can access all functionalities of Selenium WebDrive in an intuitive way. In this article. We will introduce how to use Selenium to login RAVE EDC via Chrome and fill all required information programmatically. This will allow us to perform the download operation according to the real-time demand.

INTRODUCTION

Python is an interpreted, object-oriented, dynamic data typed high-level programming language.Python provides efficient high-level data structures, as well as simple and efficient object-oriented programming.Python syntax and dynamic typing, as well as the nature of the interpreted language, make it a programming language for scripting and rapid application development on most platforms. As versions continue to be updated and new features are added to the language, it is increasingly being used for independent, large project development.

Selenium is a suite of tools for automating web browsers. At the core of Selenium is WebDriver, an interface to write instruction sets that can be run interchangeably in many browsers. Once we have installed everything, only a few lines of code get we inside a browser. We can find a more comprehensive example in Writing our first Selenium script.

Here, below example introduces how to open Chrome browser and search for "weather" in Baidu through Selenium Python：

```
from selenium import webdriver
import time
from selenium.webdriver.common.keys import Keys
driver = webdriver.Chrome(r'C:\<user>\chromedriver')
driver.get("http://www.baidu.com/")
elem = driver.find_element_by_name('wd')  # Find the search box
elem.send_keys('weather' + Keys.RETURN)
time.sleep(3)
driver.close()
```

In this example, we use the .find_element_by_name() method, which searches for the element name within the input HTML tag. And we can also search for this term using other methods.

- CSS ID: .find_element_by_id(“id-search-field”)
- DOM Path: .find_element_by_xpath(“//input[@id=’id-search-field’]”)
- CSS class: .find_element_by_class_name(“search-field”)

Selenium’s Python Module is build to perform automated testing with Python, which works with elements. An element can be a tag, property, or anything, it is an instance of *class selenium.webdriver.remote.webelement.WebElement*. After finding an element on screen using selenium,

we might want to click it or find sub-elements, etc. Selenium provides methods around this WebElement of Selenium. Here, we list some common methods.

Element Methods	Description
is_selected()	check if element is selected or not. It returns a boolean value True or False.
is_enabled()	check if element is enabled or not. It returns a boolean value True or False.
get_property()	get properties of an element, such as getting text_length property of anchor tag.
send_keys()	send text to any field, such as input field of a form or even to anchor tag paragraph, etc.
click()	click on any element, such as an anchor tag, a link, etc.
clear()	clear text of any field, such as input field of a form or even to anchor tag paragraph, etc.
value_of_css_property()	get value of a css property for a element.
location	get location of element in renderable canvas.

Table 1. List of some methods for WebElement in Selenium Python

For example, we can select value in a drop down element:

```
from selenium.webdriver.support.select import Select
select = Select(driver.find_element_by_name('name'))
# select.select_by_index(index)
# select.select_by_visible_text("text")
select.select_by_value(value)
```

For drop down element, Selenium Python provides more than one way to select the value.

PROCESS FOR AUTO DOWNLOAD EDC DATA IN RAVE

To download EDC data in RAVE manually, the usual step is log into RAVE and add a task. In the process of clinicl trials, EDC data downloading can happen at any time for different propose, which make us must be on standby before EDC data is ready. Like the example above, we can use code to implement the process of downloading data step by step. And then automatic EDC data download is achieved by executing this code automatically. Using Selenium Python, it is easy to do that.

STEP 1 CREATING AN INSTANCE OF CHROME

Selenium Python is functional for many browsers, prior to the start, we need to download a driver. Selenium requires a driver to interface with the chosen browser. Firefox, for example, requires geckodriver, which needs to be installed before the below examples can be run. Make sure it's in your PATH, e. g., place it in /usr/bin or /usr/local/bin.

Failure to observe this step will give you an error as below:

selenium.common.exceptions.WebDriverException: Message: 'geckodriver' executable needs to be in PATH.

Other supported browsers will have their own drivers available.

Chrome:	https://sites.google.com/chromium.org/driver/
Edge:	https://developer.microsoft.com/en-us/microsoft-edge/tools/webdriver/
Firefox:	https://github.com/mozilla/geckodriver/releases
Safari:	https://webkit.org/blog/6900/webdriver-support-in-safari-10/

Table 2. Links to some of the more popular browser drivers

Here we still work on Chrome:

```
from selenium import webdriver
from selenium.webdriver.support.select import Select

chrome_options = webdriver.ChromeOptions()
# Disable image
prefs = {"profile.managed_default_content_settings.images": 2}
chrome_options.add_experimental_option("prefs", prefs)
# Save user data
chrome_options.add_argument(r'--user-data-dir=C:\<user>')
browser = webdriver.Chrome(r'C:\<user>\chromedriver',
chrome_options=chrome_options)
```

ChromeOptions method: adding Chrome options. Disabling image and saving user data are using to save time and keep the status for log in.

Chrome method: create an instance of Chrome with the path of the driver that downloaded through the websites of the respective browser. The instance of Chrome is characteristic of Object-oriented programming (OOP) in Python, which is a method of structuring a program by bundling related properties and behaviors into individual objects. Now we have a Chrome wedrive object named browser, and can control web using the attributes and methods of this object, like accessing a webpage.

STEP 2 LOG INTO RAVE

After obtaining the instance of Chrome, we can perfrom any operation, just like opening Google manually:

```
browser.get("https://login.imedidata.com/login")
username = "username"
password = "password"
search_bar = browser.find_element_by_name("session[username]")
search_bar.clear()
search_bar.send_keys(username)
search_log = browser.find_element_by_name("session[password]")
search_log.clear()
search_log.send_keys(password)
browser.find_element_by_id("create_session_link").click()
```

Finding the textbox for ‘username’ and ‘password’, and then enter username and password is main function in above code. It is just like a simple Artificial Intelligenece, because we do not need to find the textbox manually that have done by code.

STEP 3 SCHEDULE A TRANSFER TASK

After accessing to imedidata, the next step is enter the message for schedule in RAVE. We need to find element attributes, like name and id by checking the source code of the web page. The key point of this step is find the correct elements what we want. It is the basis for automatic control of web pages. And next, we need to perform a series of operations on these elements step by step, just as mannul actions on a web page.

STEP 4 CLOSE CHROME

When we saved the schedule page, the final step is close and quit browser:

```
browser.close()
```

OTHER METHODS FOR THE INSTANCE OF BROWSER

.title methods: Once the page loads successfully after *.get()* method, we can use the *.title* attribute to access the textual title of the webpage. If we wish to check whether the title contains a particular substring, we can use the assert or if statements. For simplicity, let us print the title of the page.

.current_url method: Sometimes, actions on some elements will trigger a change in the URL with the search results in the window. To confirm the current URL of the window, we can use this command.

.switch_to_window() method: Our web application may require us to work with multiple windows and frames. Common use cases of working on new windows are social logins and file uploads. The *.switch_to_window()* method of the driver will help us to change the active window and work on different actions in a new window.

.window_handles method: return a list of all current window names.

.switch_to_default_content() method: switch focus to a frame within a window through the *.switch_to_frame()* method. To switch back to the primary window after completing relevant actions, we can use this method.

CONCLUSION

In the process of clinical trials, there are many complex and repetitive manual works. Using programs to realize automation, the automation not only saves time but gets cost benefits too. This paper introduces the general process on auto download EDC data in RAVE by control Chrome. This greatly improves the efficiency of daily work and reduce manual errors. There are still many more steps that can be automated to further improve usability. For example, we can read specs to batch add sechedule task or auto send email to user after completed. We would be very happy to hear from any talented experts in pharmaceutical industry to explore more efficient methods, hope this paper inspire you to develop more automate tools for our daily work.

ACKNOWLEDGMENTS

I would like to acknowledge Jie Liu who helped me on this logic and programming.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Zhouhao Pan
BeiGene
zhouhao.pan@beigene.com