Sahana Eden Regression Test Platform

Table of Contents

Introduction

The test suite is run using Selenium RC which supports a broad collection of browsers. The browser selected will be opened up in its own window and it is possible to see the navigation of the web site.

The formatting of the results is done using HTMLTestRunner and extension of the unittest TestRunner.

The regression tests are broken down into modules. Wherever possible the modules should be designed such that they can be run independently of other modules. There are several important exceptions to this as follows:

- **CreateTestAccount** This module will create the user testing@example.com this is a special account created with administration rights and can be used by any other module that requires an account with administration rights. Obviously because of this the module is not strictly a testing module rather a utility module but is still runs through Selenium RC. To create an account with admin rights this module needs to know the details of a user with admin rights and so this (and only this) module will require that information. All other modules should use the testing@example.com user created by this module. If required this module should be run first.
- **DeleteTestAccount** This module is used to delete the testing@example.com user created by the *CreateTestAccount* module. If run this should be the last module run. If it is envisaged that lots of testing will be done (such as in the process of developing new test modules) then an appropriate course of action would be to run the *CreateTestAccount* module once and then don't run the *DeleteTestAccount*; this means that the testing@example.com user will be created and then not deleted, for all subsequent runs it will not be necessary to run the *CreateTestAccount*. Once the testing has been completed the *DeleteTestAccount* can be run to tidy up the user list.
- **UserManagementCreate** This module is similar to the *CreateTestAccount* except it should be used to create non administrator users that are required by a testing module. To add extra users to this see the file **selenium/data/user.txt**.
- **UserManagementDelete** This module is similar to the *DeleteTestAccount* except that it will delete all the users created by the *UserManagementCreate* module.

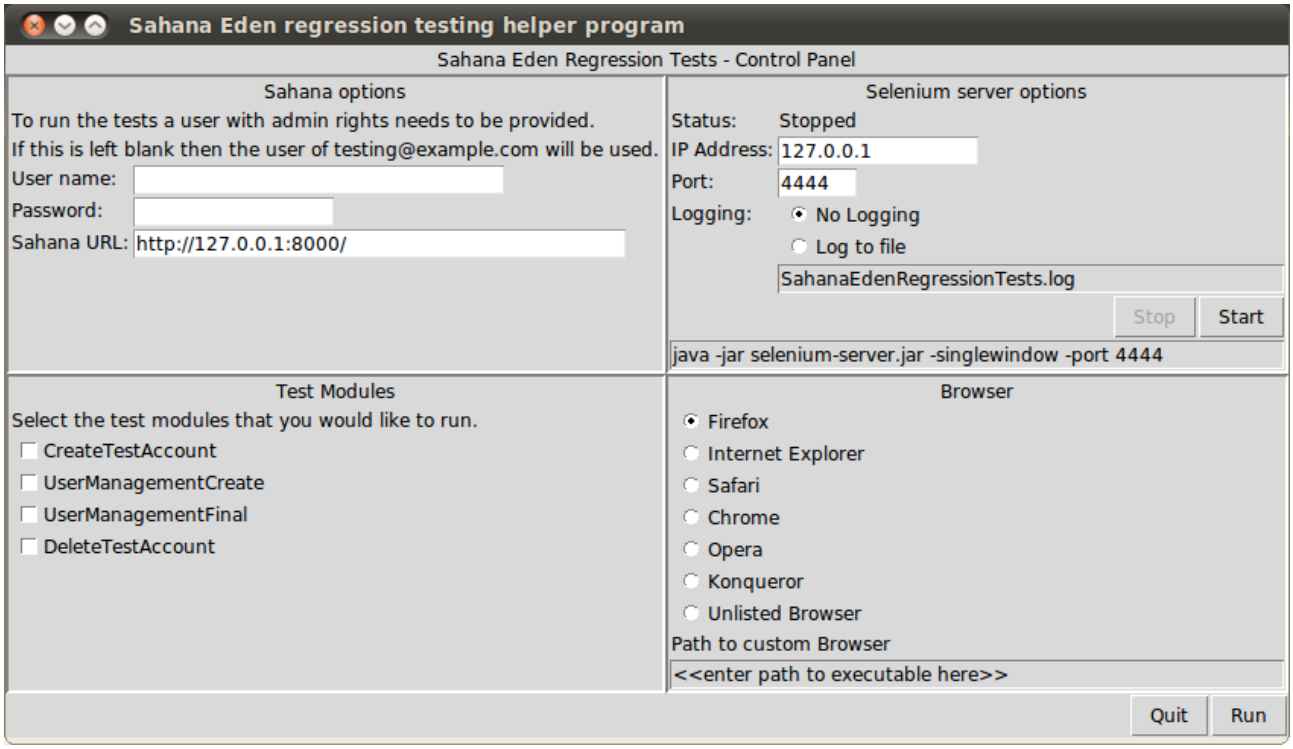
Locating the code

All of the code resides in the web2py/applications/eden/static/selenium directory. At this level there are five directories and one file as detailed blow:

- **client** This is the code from Selenium for the Python client included with Selenium RC
- **data** This holds data files that will be read by the python scripts to help automate the testing process.
- **results** This hold the results in an html file from the last run
- **scripts** This holds all of the python code for the tests, including utility methods and the GUI.
- **server** This is the code from selenium for the remote control server
- **seleium-version** text file describing the version of selenium server and client that is currently installed. This is used to keep track of releases by the Selenium project. Additionally it keeps version information on the HTMLTestRunner class that is used.

Running the Regression Tests GUI

A simple GUI has been developed to assist with the running of the tests. This is located in the selenium/scripts/regressionTests.py file.



The window is subdivided into four parts as follows:

Sahana Options

- This consists of the user name and password. These are only needed if the test module CreateTestAccount is being run.
- The URL to Sahana is required and a default value is given. The testing suite assumes that this is running if it is not then the testing suite fails.

Test Modules

- This is a list of modules these are generated dynamically from the selenium/data/testModules.txt file

Selenium server options

- This attempt to see if the Selenium server is running and helps with starting it. This only works if the GUI is being run on the same machine as Selenium RC. This works well under Linux, and moderately under Windows.
- The IP address and port are set to the default values, and logging is not enabled by default.
- The server can be started or stopped by using the appropriate buttons.
- The command to start the server is given below the buttons. So if this doesn't work on a particular configuration report the problem and use the command on a terminal to continue working.

Browser

- Selenium RC has native support for a wide range of browsers many of which are listed in this panel. If there is demand for a non-listed browser then select the unlisted browser and add the path to the browser. Support for this browser my exist so it may worth requesting for this browser to appear in the GUI if it is likely to be used a lot. The list of browsers I held in the selenium/data/browser.txt file.

Writing new Tests

The tests are all in Python but the best way to get started is to run Selenium IDE which creates an HTML file that can run these tests. These tests can then be converted to Python and the code can then be modified to use features such as functions, conditions and loops which are not available in HTML.

Currently the Selenium IDE is only available in Firefox but in conjunction with Firebug to locate the appropriate elements this forms a powerful environment to develop the initial tests.

The Test Case

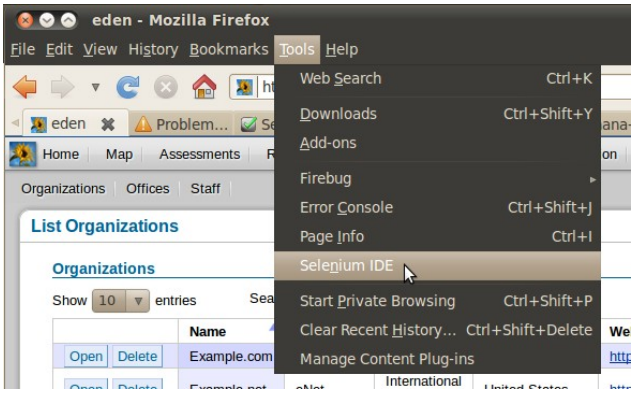
The following sections will illustrate the process of creating a test case. It will step through the test given in the test cases for the organization registry. An example is shown below:

TC	Use Case / Module	Feature/Action Tested	Prerequisite	Test Scenario Description	Input Data	Expected Result	Test Results
1	Organization Registry	List Organization	User has logged to the "Sahana Eden Disaster Management System" & navigated to the "List Organizations" page	"1.Verify the ""Search"" functionality 2.Verify the buttons in the page 3.Verify the Links in the page 4.Verify the Drop down menus 5.Verify the UI features of the page "	N/A	"1.Search functionality is functioning properly 2.All the Buttons are working properly 3.Links are navigate to the relevant page 3.1 Display the relevant web page 4.Display the relevant values under drop down menu 5.UI features are formatted accordingly	Pass
2	Organization Registry	Add Organization	User has logged to the "Sahana Eden Disaster Management System" & navigated to the "Add Organizations" page	"1. Verify the buttons of the page. 2. Verify the Links of the page 3.Verify the drop down list are working on Add Oraganization page 4.Verify the help links are working 5. Verify the UI features of the page "	N/A	"1.All the Buttons are working properly 2.Links are navigate to the relevent page 2.1.UI features are formatted accordingly on Navigated page 3.Display the relevent values under drop down menu 4.Display the help tips 5.UI features are formatted accordingly "	Pass

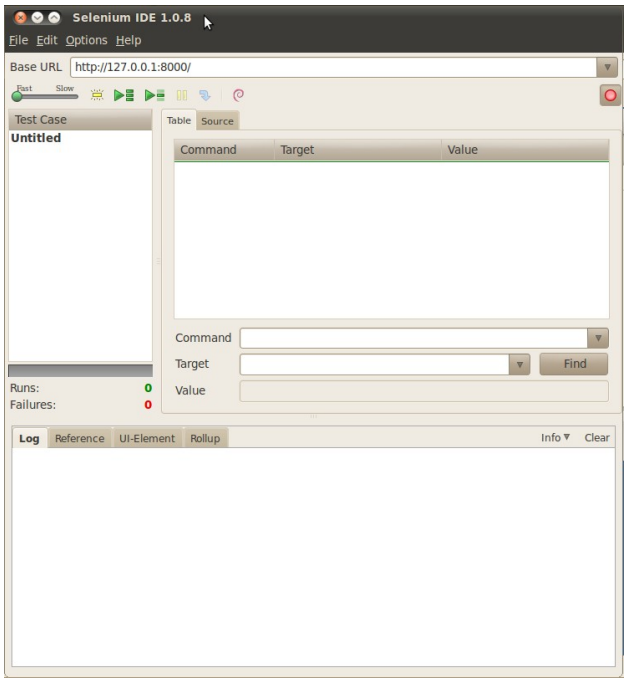
Creating a test case using Selenium IDE

To start building the test suite this section will use the Selenium IDE to record the actions required to add an organisation to the registry.

1 Start the Selenium IDE by selecting Tools → Selenium IDE from within Firefox



2 A blank Selenium IDE window will be opened



3 Now switch from Selenium IDE to the web browser and put the instance of eden in a "clean state". Log out, go to the home page and then return to the Selenium IDE.

4 Selenium IDE will record all of the actions that you perform on the target website. If you find that the IDE has recorded what you have just done the over the commands right click and select Clear All.

4.1 Set the base URL to the address of eden such as: http://127.0.0.1:8000

4.2 Click on the red record button, if it is not already recording

5 Now switch from Selenium IDE to the web browser and select Organisation Registry from the menu. Then select Organisations → Add. Look at the Selenium IDE to ensure that the actions already performed are being recorded, it should look like this:

Table Source

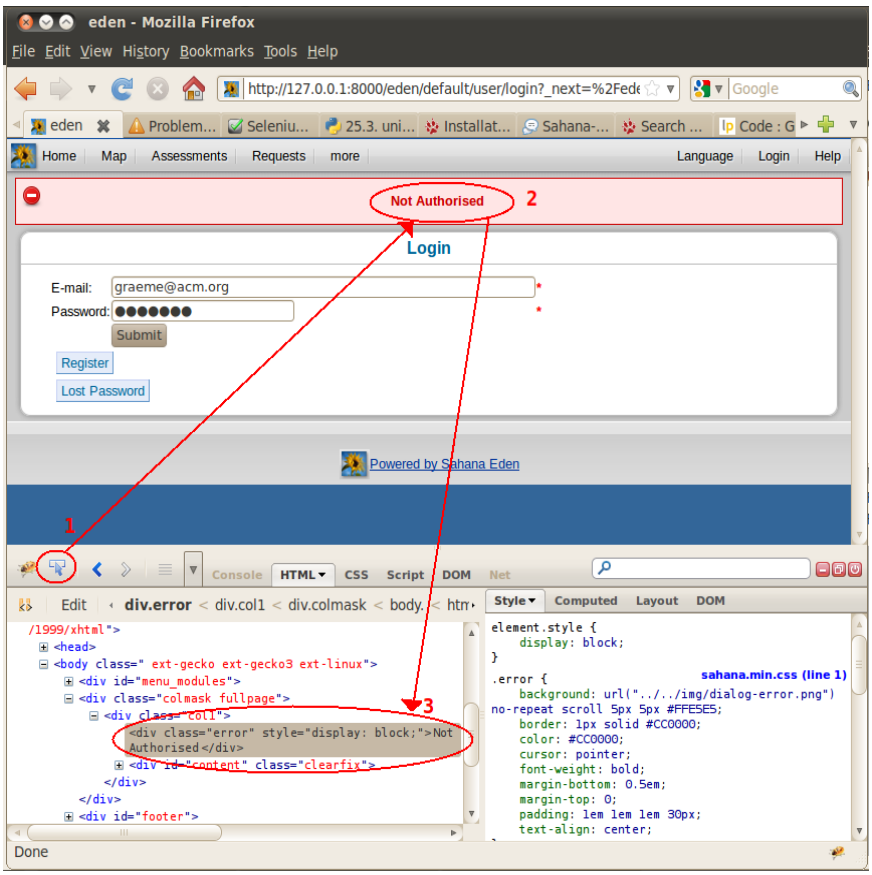
Command	Target	Value
open	/eden/default/index	
clickAndWait	link=Organisation Registry	
clickAndWait	link=Add	

6 Meanwhile on the web page a warning message is displayed stating that we are not Authorised to perform this action and we are on the login page. This will become our first check that the test is working correctly. Open Firebug and inspect the page to find the HTML tags on this page.

6.1 Click on the inspect icon

6.2 Click on the "Not Authorised" text to reveal the HTML behind this page

6.3 Look at the firebug panel to see that the HTML element is held with a div element with a class of "error"

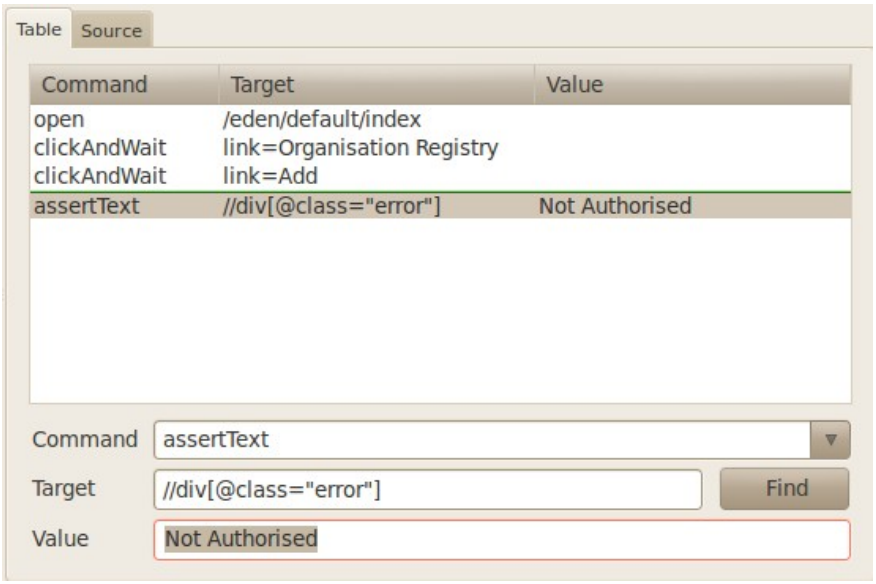


7 Move back to the IDE and in the command area select the blank line below the existing commands:

7.1 Add the command **`assertText`**, notice as you type options are provided to you. Once you have entered the command the details of this command are given in the Reference tab below.

7.2 Set the target to be **`//div[@class="error"]`**, you can check that this is correct by pressing the locate button, the html element should flash, otherwise an error message will appear in the log tab.

7.3 In the value edit box add **`Not Authorised`**



7.4 Right click on the command that has just been built and select **`Execute this command`**. The command should run successfully and the command should now be

highlighted in a pleasing green colour. If you get any error check the error log and correct the command.

8 Repeat this process to check that the Login page is showing: Remember to select a new command by clicking the area below the last command. By using Firebug you should see that the Login element is a H2 tag. Because this is the only tag of its type on the page you can use that at the location, //H2

9 After adding two assert commands we can return to recording the action by going to the web page and logging in. For the moment add your own login detail later we will see how by using a Python function one of the users specifically created for for testing will be used. Ensure that the IDE is recording click in the email box and type the email, next tab to the password and enter the password for this used and finally press the Submit button. This should have resulted in a **`Logged in`** message and the **`Add Organization`** page being displayed. Time for some asserts

9.1 Add an assert to check that the Logged in message is being displayed

9.2 Add an assert to check that the Add Organisation heading is being displayed

9.3 Note: you can copy and paste the earlier asserts and modify them as appropriate, just realise that the Logged in message is not in the error **`div`** but a confirmation **`div`**.

10 Once more check that the record button is on and return to the web page to start entering the details of the organization. Once the details have been entered and saved time for two more asserts, **`Organisation added`** and **`List Organizations`**.

At this stage all the commands have been entered to add an organisation and this script can be saved and then run. Before running the tests, delete the organization just created and log out. Then from the IDE click the **`Play Current Test Case`** button.

When I was running the test I found that there was a delay between the page loading and the error and confirmation messages appearing on the screen. To fix this I added an Selenium command pause 1000. The 1000 must be entered in the target edit box (not the value, i.e. the second not third edit box) and I copied this before each assert on the error or confirmation **`div`**.

Selenium and Python

From the IDE select the source tab and this will show the underlying HTML script that Selenium uses. Select Options → Format → Python - Selenium RC and the HTML will be converted to Python. This can now be copied and pasted into a new Python file for this test case.

Create a file called **`organisation.py`** in the selenium/scripts/ directory and paste the code in. Make sure that the test organisation has been deleted and the Selenium RC server is running run this script from within the IDE. If the test didn't work there should be a trace explaining where the test failed.

Assuming that the test worked, the code is now ready for modification. The first change will be to log in using the **`Action.login()`** method.

- Add the line **`import actions`**
- In the **`setUp()`** method of the OrganizationTest class add the following line:

```
self.action = actions.Action()
```

- Replace the lines that perform the login with a call to the login() method

Lines removed

```
sel.click("auth_user_email")
sel.type("auth_user_email", "graeme@acm.org")
sel.type("auth_user_password", "<<<password removed>>>")
```

```
sel.click("//input[@value='Submit']")
sel.wait_for_page_to_load("30000")
time.sleep(1)
self.assertEqual("Logged in",sel.get_text("//div[@class=\"confirmation\"]"))
```

Replaced with

```
self.action.login(self, "testing@example.com", "testing" )
```

- The login function will actually move to the log in page whilst in the test case we have just created we went there by default, due to not having sufficient authorisation. So it is necessary to add one more line to move to the add organisation page. This can be done in one step as follows:

```
sel.open("eden/org/organisation/create")
```

Now run the test, should you have any errors it should be a simple case to correct them. Note that this assumes that you have the testing@example.com user created, to do that run the GUI with just the **`CreateTestAccount`** test selected (and add a admin user name and password).

The modified code should look like this:

```
from selenium import selenium
import unittest, time, re
import actions

class OrganisationTest(unittest.TestCase):
    def setUp(self):
        self.verificationErrors = []
        self.action = actions.Action()
        self.selenium = selenium("localhost", 4444, "*chrome", "http://127.0.0.1:8000/")
        self.selenium.start()

    def test_organzation(self):
        sel = self.selenium
        sel.open("/eden/default/index")
        sel.click("link=Organisation Registry")
        sel.wait_for_page_to_load("30000")
        sel.click("link=Add")
        sel.wait_for_page_to_load("30000")
        time.sleep(1)
        self.assertEqual("Not Authorised", sel.get_text("//div[@class=\"error\"]"))
        self.assertEqual("Login", sel.get_text("//h2"))
        self.action.login(self, "testing@example.com", "testing" )
        sel.open("eden/org/organisation/create")
        self.assertEqual("Add Organization", sel.get_text("//h2"))
        sel.type("org_organisation_name", "Example.com")
        sel.type("org_organisation_acronym", "eCom")
        sel.select("org_organisation_type", "label=Private")
        sel.click("//option[@value='10']")
        sel.select("org_organisation_cluster_id", "label=Logistics")
        sel.select("org_organisation_country", "label=United Kingdom")
        sel.type("org_organisation_website", "www.example.com")
        sel.click("//input[@value='Save']")
        sel.wait_for_page_to_load("30000")
        time.sleep(1)
        self.assertEqual("Organization added",
sel.get_text("//div[@class=\"confirmation\"]"))
        self.assertEqual("List Organizations", sel.get_text("//h2"))

    def tearDown(self):
        self.selenium.stop()
        self.assertEqual([], self.verificationErrors)

if __name__ == "__main__":
    unittest.main()
```

Next we will remove the command that create an organisation and put them in a function, then call this function twice so that two organisations can be created. Which will look as follows:

```
    def create_organisation(self, name, acronym, type, cluster, country, website):
        sel = self.selenium

        name = name.strip()
        acronym = acronym.strip()
        type = type.strip()
        cluster = cluster.strip()
        country = country.strip()
        website = website.strip()

        sel.open("eden/org/organisation/create")
        self.assertEqual("Add Organization", sel.get_text("//h2"))
        sel.type("org_organisation_name", name)
        sel.type("org_organisation_acronym", acronym)
        sel.select("org_organisation_type", "label="+type)
        sel.click("//option[@value='10']")
        sel.select("org_organisation_cluster_id", "label="+cluster)
        sel.select("org_organisation_country", "label="+country)
        sel.type("org_organisation_website", website)
        sel.click("//input[@value='Save']")
        sel.wait_for_page_to_load("30000")
        time.sleep(1)
        self.assertEqual("Organization added",
                         sel.get_text("//div[@class=\"confirmation\"]"))
        self.assertEqual("List Organizations", sel.get_text("//h2"))
```

With the function calls as follows:

```
        self.create_organisation("Example.com", "eCom", "Private", "Logistics", "United
Kingdom", "www.example.com")
        self.create_organisation("Example.net", "eNet", "International NGO", "Recovery",
"United States", "www.example.net")
```

This can now be modified to use data driven testing by creating a text file with the organisation data in that file. Create a file called **`organisation.txt`** in the **`selenium/data`** directory with the following contents:

```
Example.com, eCom, Private, Logistics, United Kingdom, www.example.com
Example.net, eNet, International NGO, Recovery, United States, www.example.net
```

Then change the **`test_organzation()`** method to read this file as follows:

```
        source = open("../data/organisation.txt", "r")
        values = source.readlines()
        source.close()
        for org in values:
            details = org.split(',')
            if len(details) == 6:
                self.create_organisation(details[0].strip(),
                                         details[1].strip(),
                                         details[2].strip(),
                                         details[3].strip(),
                                         details[4].strip(),
                                         details[5].strip(),
                                         )
```

Now all that is needed to add another organisation to the test is to add a line to the **`organisation.txt`** file.

This test can be run from with the IDE but to allow it to run from the GUI the file **`testModules.txt`** needs to know about the new module, which is done with the following line:

```
Organisation, organisationTest
```

The first element is the value that will be displayed in the GUI the second is the name of the file.

Modifying the GUI

Outstanding tasks

- GUI - Remember the selection made by the user so that the next time the suite is run the details are preserved (however, don't store user name or password).
- GUI - Improve the control of the Selenium server under Windows.
- GUI - Improve the control over selecting and deselecting the test modules (and ensure this is integrated with the resumption of the previous settings).
- GUI - Improve the layout of the test modules (probably add two columns to allow more modules to be added)
- GUI - add an option to keep the browser open
- Investigate and propose a code coverage mechanism.
- Trial and proposes changes for the integration of Selenium 2, currently in alpha.