**An introduction to Selenium Page Object Framework**

|  |  |  |  |
| --- | --- | --- | --- |
| Author Name | Date | Version | Comments |
| Jinesh M.D | 03/10/2010 | 1.0 | Initial Draft |
| Jinesh M.D | 03/12/2010 | 1.1 | Updated to reflect the new @SeleniumUITest annotation usage |
| Jinesh M.D | 03/30/2010 | 1.2 | Added a brief description of how framework works |
| Jinesh M.D | 11/16/2010 | 1.3 | Added new section Skipping the Tests |

**System requirements:**

1. JDK 1.6 or higher
2. Eclipse 3.5 or higher
3. Apache Ant 1.6 or higher
4. Selenium java client driver and selenium server 1.0 (included in lib)
5. TestNG jdk15 or higher(included in lib)

**Framework Structure:**

Framework

Library

Actions

Pages

Navigations

Tests

Utils

Config

Docs

Lib

Build,xml

**How to start with the Page Object Framework?**

**1. Defining the Pages:**

1. The users should define a page class for each of the web pages in the application. For example LoginPage, HomePage, HistoryPage, SummaryPage etc..
2. The Page class should be a sub class of SeleniumPage from framework, since the SeleniumPage class defines all the selenium methods so that the test need not call the selenium object and use them in each page.
3. Users should define actions for each web elements in the page. Say example the page has 3 elements a link, dropdown list, text field so the user should define 3 pageactions

* clickOnLink()
* selectDropdown(“objected”);
* typeTextField(“objected”, “inputText”);

Need call the click, type… methods defined in the base class SeleniumPage.

1. All the page classes should be placed under library->pages.

**2. Defining the Navigations:**

1. Navigation is a method which will traverse into different pages to get the desired page. For example user want to see the backup history page, for this user should navigate to Reports page and then in to backup reports page the finally backup history page. In such cases all these clicking on different links will be summarized in to a methods and form a single method which will take (navigate) to the desired page.
2. The navigation methods should start with “view”, Ex. viewSummaryPage();
3. There should be a common navigation class for each module/feature, the navigation methods should be public static final.
4. All the navigation classes should be placed under library->Navigations

**3. Defining Actions**

1. Actions are compound page actions, For example to login to the application we will be using 3 pageaction methods from LoginPage
2. typeUsername(username)
3. typePassword(password)
4. clickOnLogonButton()

But we can put these pageactions to a common method which will look like this,

logon(username, password){

logingpageobject.typeUsername(username)

logingpageobject.typePassword(password)

logingpageobject.clickOnLogonButton()

}

1. Like navigations the actions also should be grouped together by feature or component level.
2. All the action should be public static and final
3. All the action classes should be placed under library->actions.

**4. Defining the locators.xml files**

1. Actual Selenium element locators are not easy to understand(Ex: xpaths - //a[@href=sm\_xyz123]) such cases give a logical name(“summarylink” ) and place it in an xml file.
2. Users can place their locators.xml in each page repositories,
3. It’s advisable to create locators.xml in module wise instead of keeping a common locators.xml
4. The convention for the gui\_locator id is “pageclassname.locatorlogicalname” Ex: <SupportcenterLoginPage.logonbutton>1</SupportcenterLoginPage.logonbutton>
5. All the locator files will be loaded to the memory while the test initialization. (May implement dynamic loading based on request.)
6. The SeleniumPage class will be looking for the locator references.

**5. Customizing the Framework using Config files**.

1. Framework and testing features are configurable
2. All the config files should be placed under “.\config” directory
3. Framework will load them automatically when a test is initialized.
4. Framework level properties are defined in “frameworkconfig.xml” and Application specific test configuration properties are defined in “testconfig.xml”
5. All the Properties can be accessed across the framework using PropertyPool, when the test started all the properties will be loaded to the memory.
6. There is no DTD defined now, the user can create a property by defining a tag and giving its value. The tag name will be considered as Property Name, and tag value will be the property value.

Ex: <browser>iexplorer</browser>

Frameworconfig.xml (Important properties)

* Seleniumserverhost - where the selenium server is running
* Seleniumserverport – port where the selenium server running by default:4444
* Locatorbasedir – relative path from source directory where the pages package/folder resides.
* Maxexecutiontime – maximum execution time in minutes for a test to complete if a test exceeds this time limit then the framework will release the test.
* Firefoxprofilepath – Custom Firefox profile path in case we use it for downloading the files.
* Buildname – Test Build name for the report(Subject of the report mail), will be obsolete when we start using Hudson
* Testgroups – testNG groups for the execution.

testConfig.xml (Important Properties)

* + protocol – application protocol (http or https)
  + hostname – application host name (Ex:ironmountain.com)
  + port – application port (8080, 443 etc..)
  + appendurl – url after the port (ex, ssws/faces.jsp etc)

We can have customized properties for different applications, please refer the testconfig.xml file, and the “section 6. Starting with the Tests” to know how to use them.

**6. Starting with the Tests**

1. The framework architecture is capable to handle different application tests simultaneously, so before starting with a new application user should define a custom sub-test class for the specific application. Currently custom subclasses are in place for support center and account management web applications
2. The custom sub-classes should be placed under webuitest->tests
3. How to write a custom subtest class; just extend from the framework class SeleniumTest which will make your test as a Selenium Test case.
4. Define a custom test initialization method where we can pass the application “protocol –http or https”, application url, application port 8080, 443 etc.., if any append url for application, and test browser if required. (use the testconfig.xml file to set these values)
5. Refer the AccountManagementTest from webuitest->tests
6. Now start creating the actual tests using the sub-test class, For example to create a new Account management login test, write a test class “AccountManagementLoginTest” which is a sub-class of AccountManagementTest.
7. Basic Steps a Selenium Test Required

* To start a new selenium session with a browser (start test)
* To execute the test steps on the application. (execute test)
* Close the browser and stop the selenium session. (stop test)
* Be careful about the TestNG/Junit annotations because each Test Class only needs to call the start and stop methods once.

In our case the framework level SeleniumTest class will handle the starting selenium session and stopping selenium session tasks.

1. Now place the Annotation @SeleniumUITest, in the test class. (By placing the annotation @SeleniumUItest will make the class as a selenium test class and the framework can execute this test.

Refer sample test from tests🡪.amws “SampleAccountManagementTest”

**7. How the Framework works**

Here the steps the framework does when it found a class which is annotated with @Seleniumtest.

1. The framework will call the init() method from SeleniumTest and load all the property files from configuration files and load the locator files from page libraries. Now the properties and locators will be available from PropertyPool and LocatorPool classes.
2. Now framework will create a selenium session with the test properties from testconfig.xml
3. The test will be added to a test pool for further processing (in case parallel)
4. The selenium session will be passed to the pages, and the first page will be created and stored in a pool called PageFactory.
5. PageFactory will provide the Page Objects whenever required.
6. Once the test is completed the Test class must be calling the SeleniumTest’s stop() method.
7. Stop method will close the selenium session and will close the browser as well.
8. Stop method can also release the common resources in case if we need to close.

**8. Utilty classes**

1. Utility Classes for the framework and tests, If the user needs to add a specific functionality should write a Util class. For example we need a JDBC connection to test the database tables we should define the utility class to do that.

**9. Lib**

1. Currently the 3rd party jars are placed in the lib folder and packed with the framework. The common jars are, testNG, seleniumserver, selenium-java-client-driver etc)

**10. Running the tests using Ant Build.**

1. An Ant build file is already created and available in the framework base directory.
2. To use the build.xml Apache Ant 2.6 should be set with the class path.
3. The common tasks are already defined in the build.xml, the available tasks are

* Clean – cleaning the temporary files
* Compile – compiling the source code
* Startseleniumserver – to start the selenium server
* Startseleniumserverwith FFProfile - To start the selenium server with custom firefox profile (may not require if we are not using custom download actions or settings)
* stopseleniumserver –To stop the selenium server
* runTestBuild – to run the entire test suite by testNG groups.
* sendReport – sending an email to the audience

**11. Reports**

1. No custom reports implemented, using the TestNG’s built-in reporter.
2. If we are running the tests using build the reports will be placed in the “TestReports” folder.
3. Individual execution reports through Eclipse can be found under “test-output” directory.
4. Build.xml will create a zip file with the test result and send to the targeted users.

**12. Skipping the Tests**

In some cases you may have a functional test which is called a universal setup, say example you are creating a test user in your setup test, this user will be used to execute the remaining test cases in the test suite. In such cases if the user creation test fails obviously the dependent test cases fail by saying that the user does not exist or something like that. In this kind of cases we may need to skip the rest of the test cases if the setup test fails, No we will look in to how we are doing this with PageObjectFramework.

1. There are 2 new attributes/properties added to our SeleniumUITest annotation those are

**skipAllTestsIfFail** // Specify you need to skip all test if this test fails

**resetSkipAllTestsIfFail** // Specify if you need to reset the skip which set by previous test. (Need to specify while you run your tests for debugging)

These 2 properties will accept Boolean values true or false.

1. Use **notifySetupTestComplete()** in the end of test method to notify the setup Test is passed. If skipAlltestsIfFail is enabled (=true) in a test and then we need to call a notify method to say that the Skipper Test is completed successfully. To do this we will make a call to **notifySetupTestComplete();** method in the end of the test method.
2. Implement the Product-Base Test to make use of the SkipTest feature. Now we need to make the necessary changes to your base-sub test cases to make use of the feature. Now we have **“stopSeleniumTestEnableSkipTest()”** method to set the skipValue if the test fails. So if you want to make use of the feature, use this stop method. Please refer the **AccountManageMentTest**  (PageObjectConnected) for the implementation
3. Additionally if you want to individually skip a test say the test will fail due to a known issue and you don’t want to execute it all time until the issue got fixed. In such cases use

isSkipTest = true; as the first statement in the startTest();

Additionally you can set skipMessage = “Reason for skipping” mentioning the issue ID and description etc..(Make sure that the teardown will not be execute if you skip the test)

Q: If I have a build which has multiple suites, the test cases in the suites are independent of the failure of setups in the other suites in such case if the setup in first suite failes will it make to fail the tests in other suites as well?

A: Typically the answer is “Yes” but we need to use the feature “resetSkipAllTestsIfFail” to avoid this situation. So make sure that you have setup test cases in all the suites (Or in the first test case in the suite use resetSkipAllTestsIfFail=true which will set the skipFeature as false)

Q: How this will impact the individual execution of test cases?

A: Make sure that the state of skipValue is false. While running it will give message in the log says you have reset set this to “true” and tests will be skipped. So before re-running the suite make sure that you use resetSkipAllTestsIfFail=true in the first test.

Q: What will happen if I am not using the skip attributes

A: All will be set to default, that means false. Test will be executed as there is no skip feature.

Q: What are the current cons?

A: You need to make the changes in the “java-test class level”.