Selenium 2 Test Automation Framework

Version 3.0

TABLE OF CONTENTS

FRAMEWORK OVERVIEW

Selenium 2 (Webdriver) Framework

Setting up Selenium Automation for the first time or running it for separate web applications, you will require some bit of study to get started. You can read up a lot of documentation, experiment with scripts, and repeat the process every time your testing object changes. **Or you can simply use our Selenium 2 (Webdriver) Framework.**

This Selenium 2 (Webdriver) Framework is a set of guidelines, including coding standards, test-data handling, object repository treatment etc., which when followed will not only save precious time but will also provide additional benefits including increased code re-usage, higher portability, and reduced script maintenance cost.

Our Approach

For our framework we have used **'Data-Driven Framework'** with Selenium 2 (Webdriver) and Java Programming language.

A data-driven framework is where test input and output values are read from data files (ODBC sources, CVS files, Excel files and DAO objects) and are loaded into variables in captured or manually coded scripts.

In this framework, variables are used for both input values and output verification values. Navigation through the program, reading of the data files, and logging of test status and information are all coded in the test script.

Framework Features

- Well defined architectural design
- Less time to test large data
- Script execution in multiple environments
- Easier, faster and efficient analysis of result logs
- Communication of results
- Easy debugging and script maintenance
- Robust and stable due to error and exception handling
- 100% reliability of utility scripts, online execution, report packs

FRAMEWORK ARCHITECTURE DESIGN

Architecture forms the foundation of any software application. It should be robust enough to handle the desired functions efficiently and effectively. In this approach, the goal is to develop an application-independent reusable Data-Driven Framework that can be used directly across any application without spending any extra time on it.

In order to make all the components of the system work in sync, it is important to define the components and its functionalities, as well as the binding relationship between them.

Architecture Explanation

This Package Includes:

1. **Config** - Keeps all the configuration files such as property files
2. **InputTestData** - Has files containing input data for application
3. **OutputData** - Contains downloaded docs/images, fetched data in excel
4. **TestReports** - Contains ANT generated reports
5. **Util** - Utility package contains all generic functions & business functions such as email configuration setting and all other utilities
6. **TestLogs** - Contains log file corresponding to tests
7. **DAO** - Classes for accessing persistent storage, such as to a database
8. **Pages** - Page classes for particular pages

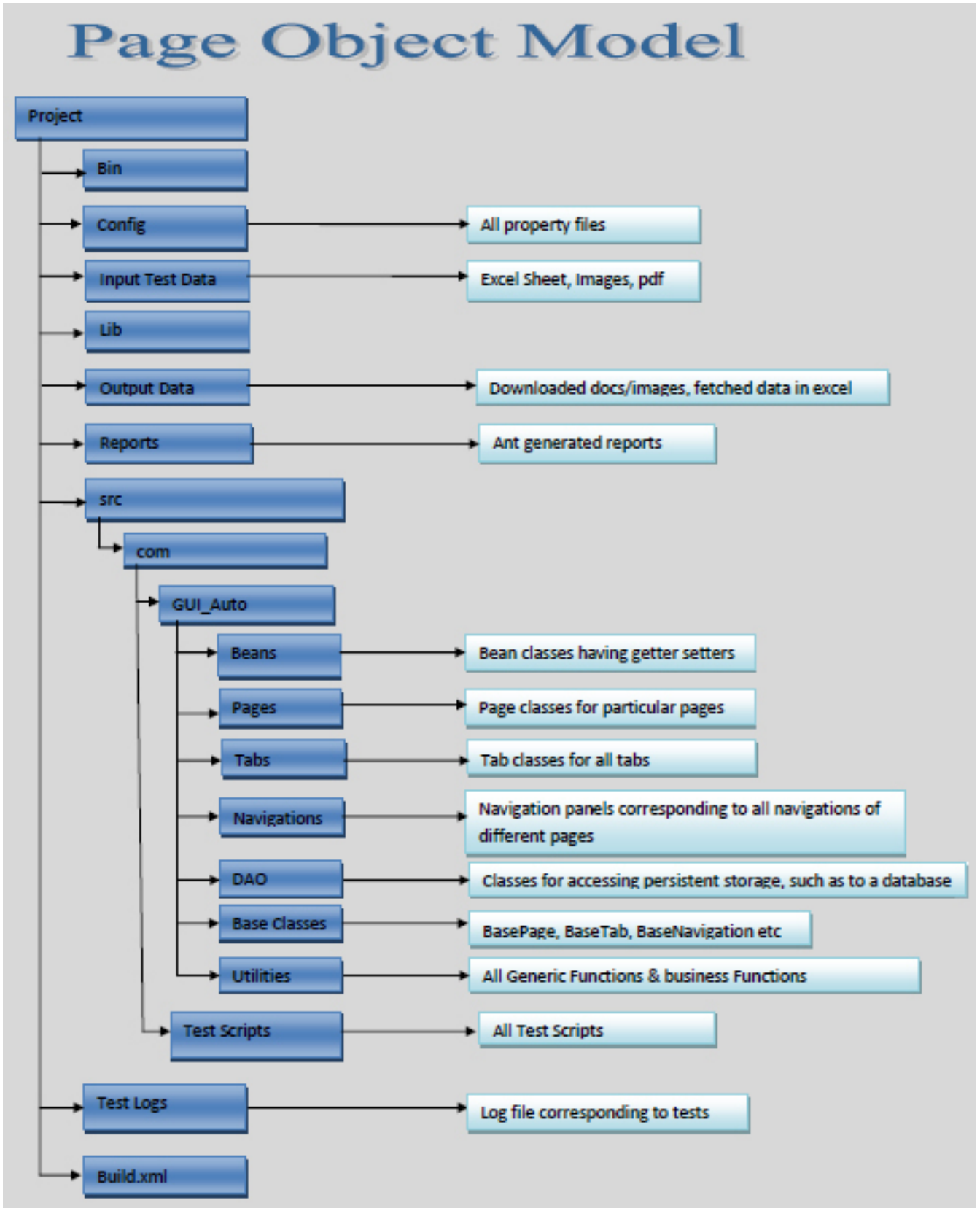

Fig1: Directory Structure of the Framework

Understanding of Page Object Model

Page Object Pattern

- A simple design pattern that models each page (or page chunk) as a class object that other classes / objects can interact with
- A very common pattern for implementing an automation framework using Selenium 2 (Webdriver)
- Classes representing pages / chunks should provide the services that a user would execute on the page

Test Script: RegistrationTest.java

Lets us take a look at our RegistrationTest:

```
public class RegistrationTest extends DBCon{
        static Logger log = Logger.getLogger(RegistrationTest.class.getName());
        private StringBuffer verificationErrors = new StringBuffer();
        DBCon dbConnection = new DBCon();
        private RegisterPage _register;
        ArrayList<UsersListBean> usersList;
        @Before
        public void setUp() throws Exception {
                super.setUp(getBrowser());
                PropertyConfigurator.configure("config/log4j.properties");
                usersList = dbConnection.loadDataFromExcel();
                _register = PageFactory.initElements(_driver, RegisterPage.class);
        }
        @Test
public void testValidRegister() throws Exception {
                assert !_users.isEmpty();
                // Open URL in browser
                _register.navigateToPageAndWait();
                // Verify that bottom navigation is present on this page
                BottomNavigationPanel bottomNav = new BottomNavigationPanel(_driver);
                assertTrue(bottomNav.isNavPresent());
                for (int i = 0; i < usersList.size(); i++) {
                        try {
                                ThankYouPage thankYouPage = new ThankYouPage(_driver);
                                // Register into the site
                                // Parameterization of different input fields - text boxes, radio button, check boxes, drop downs,
                                // file upload, etc
                                // Also parameterization of auto-suggest search drop downs
                                // Please mention the location of the file to download in the input excel sheet
                                thankYouPage = _register.validRegistration(usersList, i);
                                // Verify that welcome message appears correctly
                                assertEquals("Welcome "+usersList.get(i).getFirstName(),thankYouPage.assertUserGreeting());
                                //Verify that logout link is displayed
                                assertTrue(thankYouPage.checkLogoutLink());
                                // Verify that correct first name is displayed
                                assertEquals(usersList.get(i).getFirstName(),thankYouPage.assertFirstName());
                                // Verify that correct last name is displayed
                                assertEquals(usersList.get(i).getLastName(),thankYouPage.assertLastName());
                                // Verify that correct phone number is displayed
                                assertEquals(usersList.get(i).getPhoneNumber(),thankYouPage.assertPhoneNumber());
                                // Verify that correct gender is displayed
                                assertEquals(usersList.get(i).getGender(),thankYouPage.assertGender());
                                // Verify that correct industry is displayed
                                assertEquals(usersList.get(i).getIndustry(),thankYouPage.assertIndustry());
                                // Verify that correct country is displayed
                                assertEquals(usersList.get(i).getCountry(),thankYouPage.assertCountry());
                                log.info("User "+usersList.get(i).getFirstName()+" has registered successfully");
} catch (AssertionError ex) {
                                log.error("Value does not match", ex);
}
```

```
                        // Logout from site
                        _register.logout();
                }
        }
        @After
        public void tearDown() throws Exception {
                _driver.quit();
                String verificationErrorString = verificationErrors.toString();

                if (!"".equals(verificationErrorString)) {
                        fail(verificationErrorString);
                }
        }
}
```

Retrieving Data From DataBase:

RegisterTest extends DBCon class having function loadDataFromExcel() retrieving data from excel sheet which further extends GUI_automation_base class setting up the browser to be used as mentioned in gui_automation.properties in config folder.

Logs:

Initialize the logger for getting logs with: static Logger log = Logger.getLogger(RegisterTest.class.getName());

Setup Function:

In setup() function call the GUI_automation_base setup() function by getting the browser from gui_automation.properties in config folder, configure the log4j.properties file, load the data from excel sheet and initialize the RegisterPage instance.

Rules for Automation Framework

1. Methods performing an action that results into other page should return the page object to user. Assert and wait to load the page completely with all elements of the page.
   Example: Clicking a button on page A redirects the user to page B, then returns the object of page A
2. Naming of variables and methods:
   a. Standard JAVA naming conventions should be utilized
   b. If a method performs an action that clicks on any element on the page, the name of the method should start with click
   c. If a method retrieves any value or entity, or a list of values or entities, the method name should start with get
   d. If a method assigns a value to a web element, the method name should start with set
   e. If a method interacts with radio or check-boxes it should start with enable or disable
   f. Instance variables should be named using an underscore as a leading character. **Example:** _register, _pageName
3. Encapsulation should be used as follow:
   a. All methods intended for the implementation of tests for the automation of the web project, should be public
   b. All methods not intended for the implementation of tests should be private
   c. All variables, with the exception of enums, should be private
4. Common methods for similar web elements:
   a. Lists (HTML tables): getAll* or getAll*ByName, find*ByName, click*_ByName
   b. List-boxes: getAll*Options or getAll*Options_ByText, click*Option
5. All assertions and verifications should be within the tests using JUnit framework asserts

New Features

Selenium 2 (Webdriver) Framework Version 3 has all new features, which will make testing a whole lot easier. Key new features include:

1. **Parameterization of Different Input Types:**

   In the earlier versions, we did parameterization only for textboxes. But in version 3.0 we have also implemented it for other input types like radio button, drop downs, check boxes, auto-suggest search drop down, and file upload.

2. **Auto-suggest Search Dropdown:**

   The common Webdriver code for selecting drop down is:

```
Select gender = new Select(driver.findElement(By.id("gender")));
gender.selectByVisibleText("Female");
```

   This does not work for auto-suggest search dropdowns

   Therefore, we have used following code in our framework for these types of dropdowns:

```
driver.findElement(locator).sendKeys(value);
driver.findElement(locator).sendKeys(Keys.TAB);
```

3. **Image Comparison:**

   This feature helps you to take screenshots of images and compare them with sample images. In our code, we have added 3 sample images in the 'InputTestData' folder of the Project and compared them with the screenshots that are stored in the 'OutputData' folder.

4. **Drag and Drop Functionality:**

   Drag and drop is a very common feature. It is when you "grab" an object and drag it to a different location. We have also also added this functionality using the Actions class:

```
WebElement element = driver.findElement(By.name("source"));
WebElement target = driver.findElement(By.name("target"));
(new Actions(driver)).dragAndDrop(element, target).perform();
```

5. **Sorting Values:**

   Data sort is a common functionality used in different websites especially Business to Consumer sites. You can use framework's sorting functionality to sort values and check if the sorted order is correct or not. In the test script – SortingTest.java, we have checked sorting of table rows for three columns.

6. **Auto-Focusing On New Tab:**

   Some links on the websites open in a new tab but when we click on them via automation script, a new window opens instead of tab; and the automation script breaks. We have now added code in our framework for managing this new window using webdriver's "switchTo" method.

7. **Check and Delete Cookies:**

   Webdriver has amazing feature of cookie handling that we have used in our framework to check and delete cookies for a specific domain.

Conclusion

A robust framework helps keep pace with agile software delivery and maximize on the benefit of Selenium automation testing. It drives productivity and facilitates code reuse ultimately enhancing the quality of resulting software.