OKAPI Web UI Automation Test Framework Library

Highlights

- User-customisable (Test report/Log/Selenium Driver Options) to suit most of the needs of large organisations
- Selenium ChromeDriver, FirefoxDriver, InternetExplorerDriver, EdgeDriver and RemoteWebDriver support
- Easy to set up
- Auto-driver management
- Traditional web element locating support
- Anchor-based web element locating support for fast development
- Chain action support
- Dynamic content support (making easy to access tables, menus, and dropdowns without the need to use Selenium Select)
- Dynamic web element search for users with limited development background
- Data-driven support out of the box
- Accurate page object model (POM) auto recording for very fast development
- Reliable API to deal with most of the toughest web applications under test with minimal false positives
- Support English language. Anchor-based web element locating may not work well with web applications in non-English languages
- Easy to use

1. **User-customisable**
    a. ***Logging***
        - Users are allowed to inject their logging component via implementing **IOkapiLogger** interface
        - Comes with standard logger as NuGet package (https://www.nuget.org/packages/Okapi.Support.File/1.0.0)
    b. ***Test Report***
        - Users are allowed to customise test report format and destinations via implementing **IReportFormatter** interface. Each test action/verification point will send data to ALM, Stackify, etc.
        - Comes with standard test report as NuGet package (https://www.nuget.org/packages/Okapi.Support.File/1.0.0)

- Currently, supports reporting inline within test scripts (using attribute [TestCase] for test case, [Step] for test steps, and TestReport methods such as Verify() for verification points; and Report() at end of test cases and test steps – if needing to report steps). Future development will include a Unit test framework as NuGet which supports out of band test reporting.

```
[TestMethod]
[TestCase]
public void Dynamic_tag_and_its_text()
{
    DriverPool.Instance.ActiveDriver.LaunchPage("https://www.test.com");
    string text = Dynamic.Find("<h2> `Testing`").Text;
    void assertion() => Assert.AreEqual("Testing", text);
    TestReport.Verify(assertion);
    TestReport.Report().QuitActiveDriver();
}
```

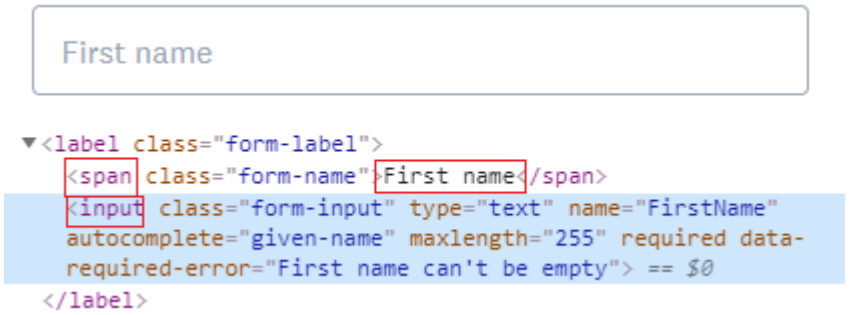
2. **Auto-driver management**

```
DriverPool.Instance.ActiveDriver.LaunchPage("https://www.test.com");
TestObject.New("//label[span[contains(text(),'First name')]]/input").SendKeys("tester");
TestObject.New(LocatingMethod.Id, "saveButton").Click();
DriverPool.Instance.QuitActiveDriver();
```

3. **Traditional web element locating support**
   - XPath, CssSelector, Id, Name, ClassName, TagName, LinkText, and PartialLinkText
   - If not specified, XPath is used by default

4. **Anchor-based web element locating support for fast development**
   - Supports two types: Anchor; TwoAnchors
   - Works based on the algorithm to find linked html tags (‘for’ attribute) and shortest DOM distance from the anchor web elements to the web element to search

Example:




```
DriverPool.Instance.ActiveDriver.LaunchPage("https://www.test.com");
TestObject.New(SearchInfo.New("span", "First name"), SearchInfo.New("input")).SendKeys("tester");
```

- 'span' can be omitted if 'First name' is unique on the page
- 'input' can be a bit more detailed to narrow down the search, using css (i.e. 'label>input')

When to use two anchors: For instance, finding a radio button (element to search) next to a label 'Yes' (anchor) under a question (parent anchor)

5. **Chain action support**

```
new TestObject(LocatingMethod.Name, "FirstName").Highlight().SendKeys("tester");
```

6. Dynamic contents support

```
var dynamicObject = TestObject.New("//label[span[contains(text(),'{0}')]]/input");
dynamicObject.SetDynamicContents("First Name").SendKeys("tester");
dynamicObject.SetDynamicContents("Last Name").SendKeys("automation");
string userNameText = TestObject.New(SearchInfo.New("span", "{0}"), SearchInfo.New("input")).SetDynamicContents("user name").Text;
```

7. Dynamic web element search


```
<button type="submit" class="btn btn-primary btn-disabled"
disabled>Get started</button>
```

```
string text = Dynamic.Find("<button> `Get started`").Text;
text = Dynamic.Find("Search tag <button> with text `Get started`").Text;
text = TestObject.New(SearchInfo.New("button", "Get started")).Text;

Dynamic.Find("Find xpath `//label[span[contains(text(),'First name')]]/input`").SendKeys("Automation");
Dynamic.Find("xpath `//label[span[contains(text(),'First name')]]/input`").SendKeys("Automation");
Dynamic.Find("I try to find xpath as `//label[span[contains(text(),'First name')]]/input`").SendKeys("Automation");
```

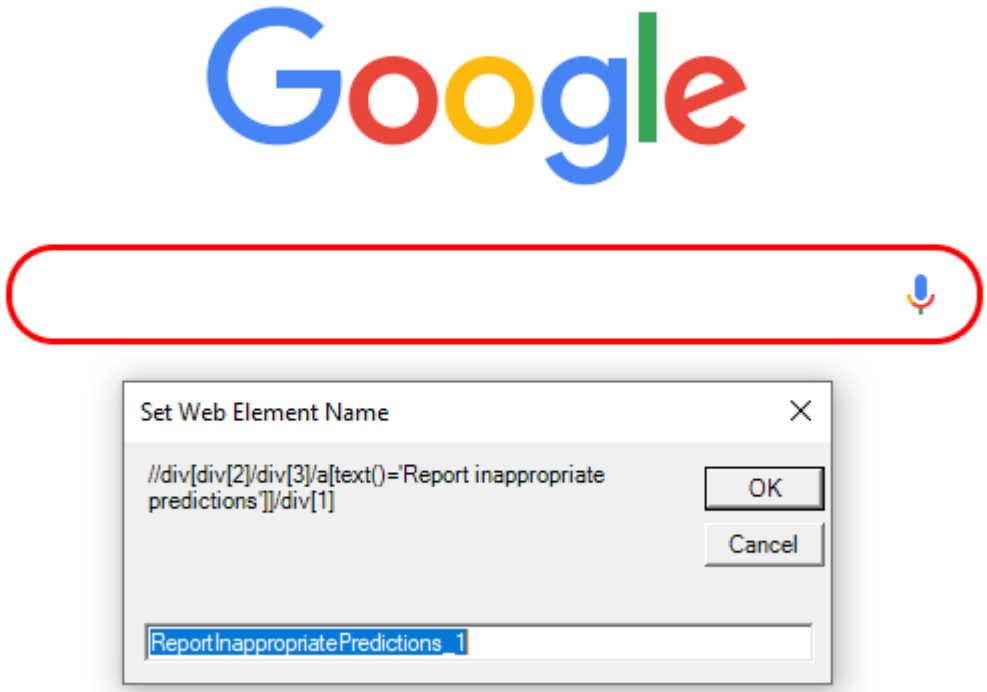
- First 3 lines do the same thing. Last 3 lines do the same thing.
- For Dynamic.Find(), tags have to be between <> and texts or locators (i.e. xpath) have to be between backticks (`)

8. **Data-driven support out of the box**
    - All the methods with Okapi dealing with data will do nothing when data passed to them as ‘null’. So simply set data as null or not will drive the interactions accordingly. Data contains business rules.

9. **Accurate Page object model auto recording for very fast development**

```
[TestMethod]
public void Codegen()
{
    DriverPool.Instance.ActiveDriver.LaunchPage("https://www.google.com");

    IList<string> usings = new List<string>
    {
        "System",
        "Okapi.Enums"
    };

    string nameSpace = "Okapi.SampleTests";

    //non-user intervention
    new CodeGen(usings, nameSpace).GeneratePOMFile("GoogleSearchPage_Generated", Util.CurrentProjectDirectory);

    //users intervention: pick which web elements (highlighted) to record and set their names or accept default names
    new CodeGen(usings, nameSpace).GeneratePOMFileUserProvidedPropertyNames("GoogleSearchPage_Generated", Util.CurrentProjectDirectory);
}
```

Below screenshot is user intervention case with web element highlighted:



Generated Page Object Model (POM) class file:

```
public class GoogleSearchPage_Generated
{
    public static TestObject Account => TestObject.New("//a[span[2][text()='Account']]");
    public static TestObject MyAccount => TestObject.New("//a[span[2][text()='Account']]/div[1]");
    public static TestObject Account_1 => TestObject.New("//a[span[2][text()='Account']]/div[2]");
    public static TestObject Account_2 => TestObject.New("//a[span[2][text()='Account']]/div[3]");
    public static TestObject Search_4 => TestObject.New("//a[span[2][text()='Search']]/span[1]");
    public static TestObject Search_5 => TestObject.New("//span[text()='Search']");
```

10. Easy to use

- Most methods are named after Selenium methods so users don't need to remember new set of method names
- Almost one main class to remember: **TestObject**
- Action and properties can be chained to reduce number of lines of code
- Overheads are kept minimal with useful utilities, including looping actions