OKAPI How To Record Web Objects

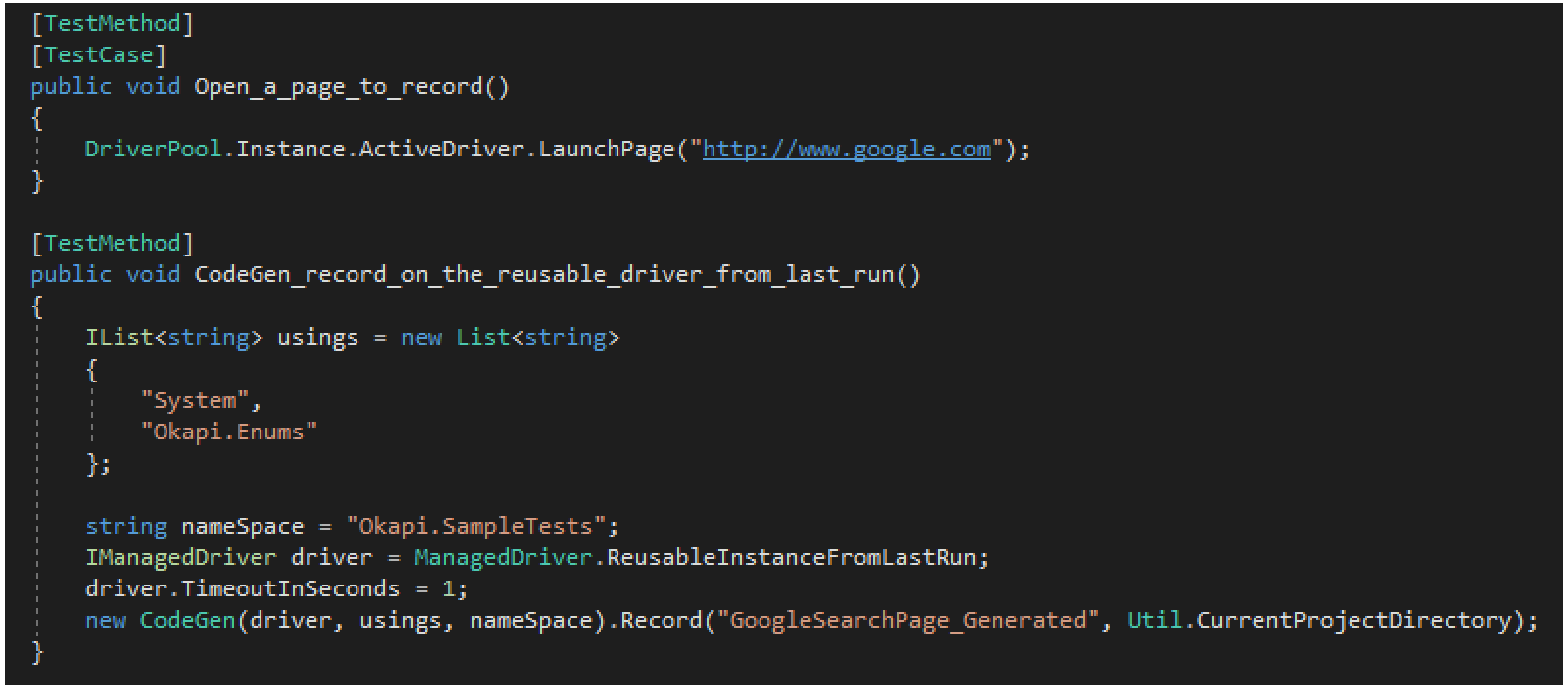
1. Okapi Object Repository

- Okapi supports recording Web UI objects into object repository under the form of Page Object Model (POM) C# classes
- After recording, each POM class contains multiple properties, each with a name and a value where the value is an Okapi **TestObject**. This test object contains a unique xpath locator for a web element in DOM
- Okapi has **CodeGen** class with methods for the recording
- The unique and advanced algorithm in Okapi will automate this process with accurate results with least refactored xpaths recorded. The xpaths recorded by Okapi will based on the own texts of html tags in the html document and the hierarchy of the html document itself. They do not depend on the tags' attributes to avoid the dynamic and auto-generated attributes by JavaScript or other web service front-end or back-end engines, which make the xpaths recorded only valid at the recording time.
- The **Record** method of the **CodeGen** class l walks users through the process and it includes two sub-processes: auto-recording, and manual-recording
- **Auto-recording**: Okapi makes the decision (in theory the outcome is 100% correct, unique and least refactored xpaths), offers unique values and default names for the recorded properties. It will highlight each web object on a web page and will allow users to change names for the recorded properties. Users have an option to ignore web objects not needed for their automation testing purposes or to stop the recording.
- **Manual-recording**: Users construct each web object locator; Okapi highlights it if found and users can save it.

Notes: there are two defects with the libraries which Okapi depends on which makes the auto-recording results not 100% correct in some cases though minor. The defects include the xpaths recorded not correct in some cases, and finding by anchor unable to find web objects in some cases such as web objects on a pop-up window cannot be found by anchor. The first bug is caused by a minor bug in ExtSelenium library which I am an author so I can fix it when needed. The second bug is caused by a bug in dcsoup library.

2. Sample Code

- Run the first test to open a Chrome driver with Google and you can manually do something on Google. Leave the Chrome driver opened after the test stops
- Run the second test to record objects on Google page

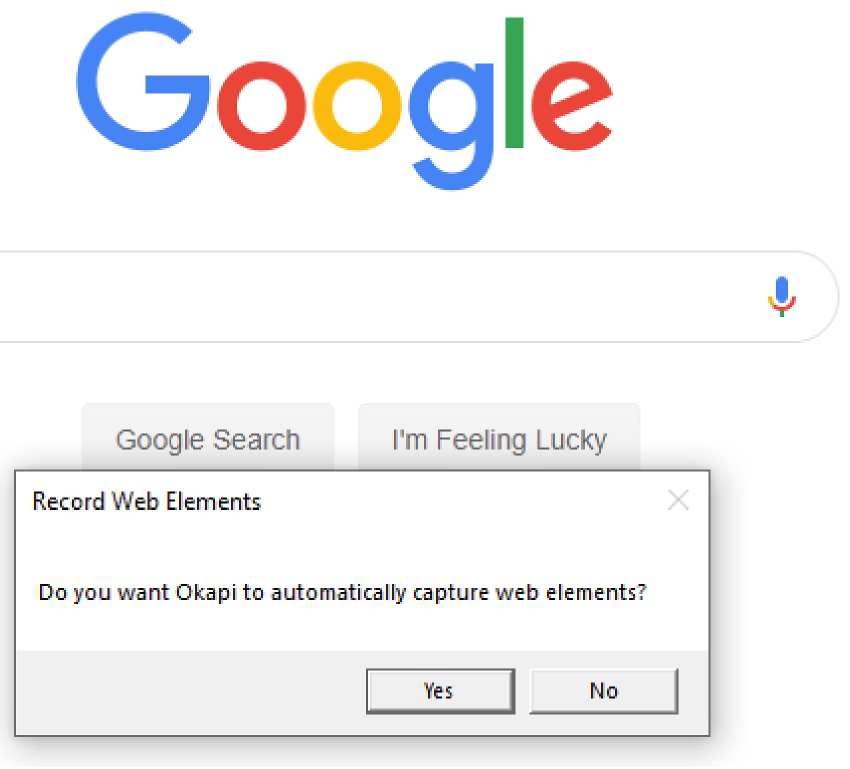


```
[TestMethod]
[TestCase]
public void Open_a_page_to_record()
{
    DriverPool.Instance.ActiveDriver.LaunchPage("http://www.google.com");
}

[TestMethod]
public void CodeGen_record_on_the_reusable_driver_from_last_run()
{
    IList<string> usings = new List<string>
    {
        "System",
        "Okapi.Enums"
    };

    string nameSpace = "Okapi.SampleTests";
    IManagedDriver driver = ManagedDriver.ReusableInstanceFromLastRun;
    driver.TimeoutInSeconds = 1;
    new CodeGen(driver, usings, nameSpace).Record("GoogleSearchPage_Generated", Util.CurrentProjectDirectory);
}
```

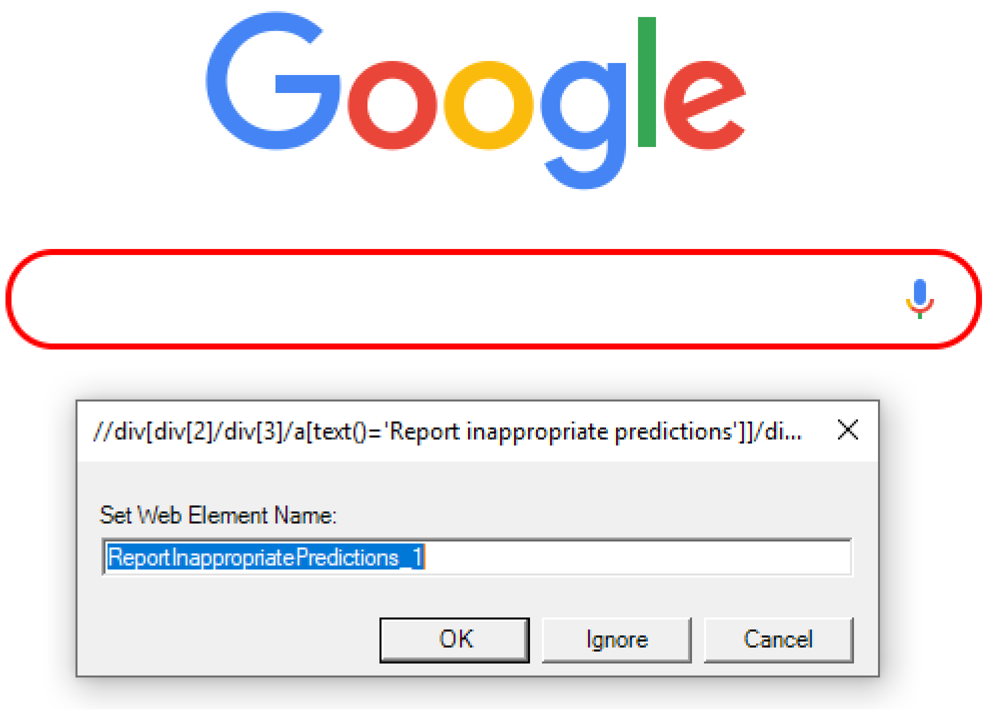
You will see:



Record Web Elements

Do you want Okapi to automatically capture web elements?

Click **Yes** to let Okapi perform auto-recording. Wait up to a minute for Okapi to calculate xpaths for all the tags in the html document of the web page displayed in Chrome driver.



//div[div[2]/div[3]/a[text()='Report inappropriate predictions']]/di...

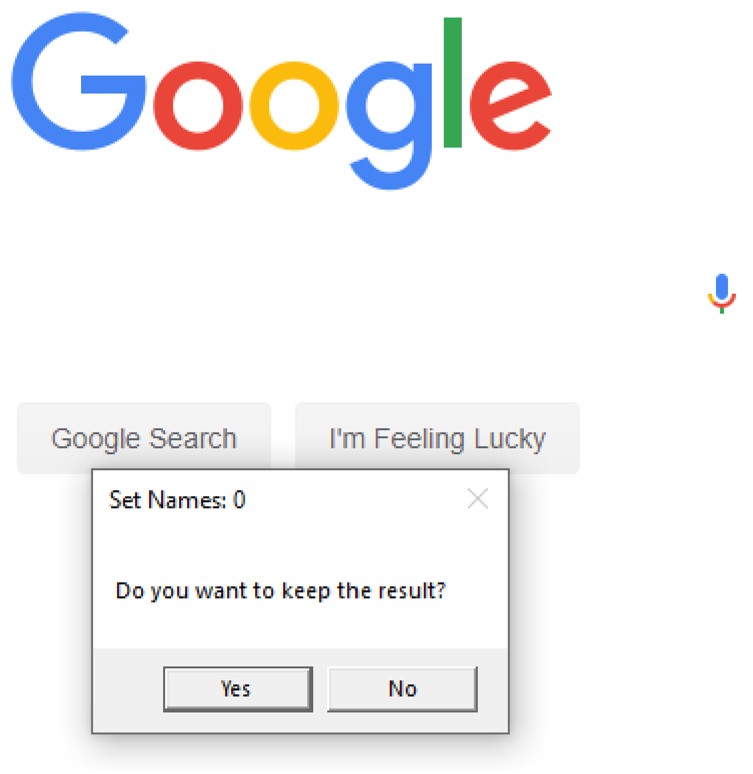
Set Web Element Name:

ReportInappropriatePredictions_1

Click **Ignore** for the highlighted objects which you don't want to save into C# POM file.

Change the names if needed and click **OK** to save the highlighted objects which you want to record.

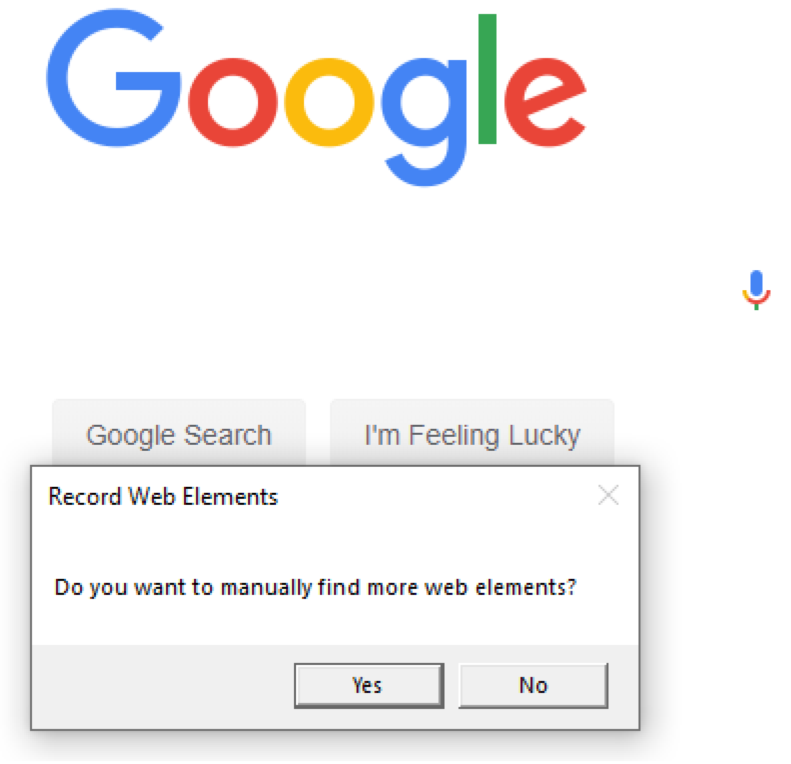
Click **Cancel** to discontinue the process. When you click Cancel, Okapi will ask you as below:



Set Names: 0

Do you want to keep the result?

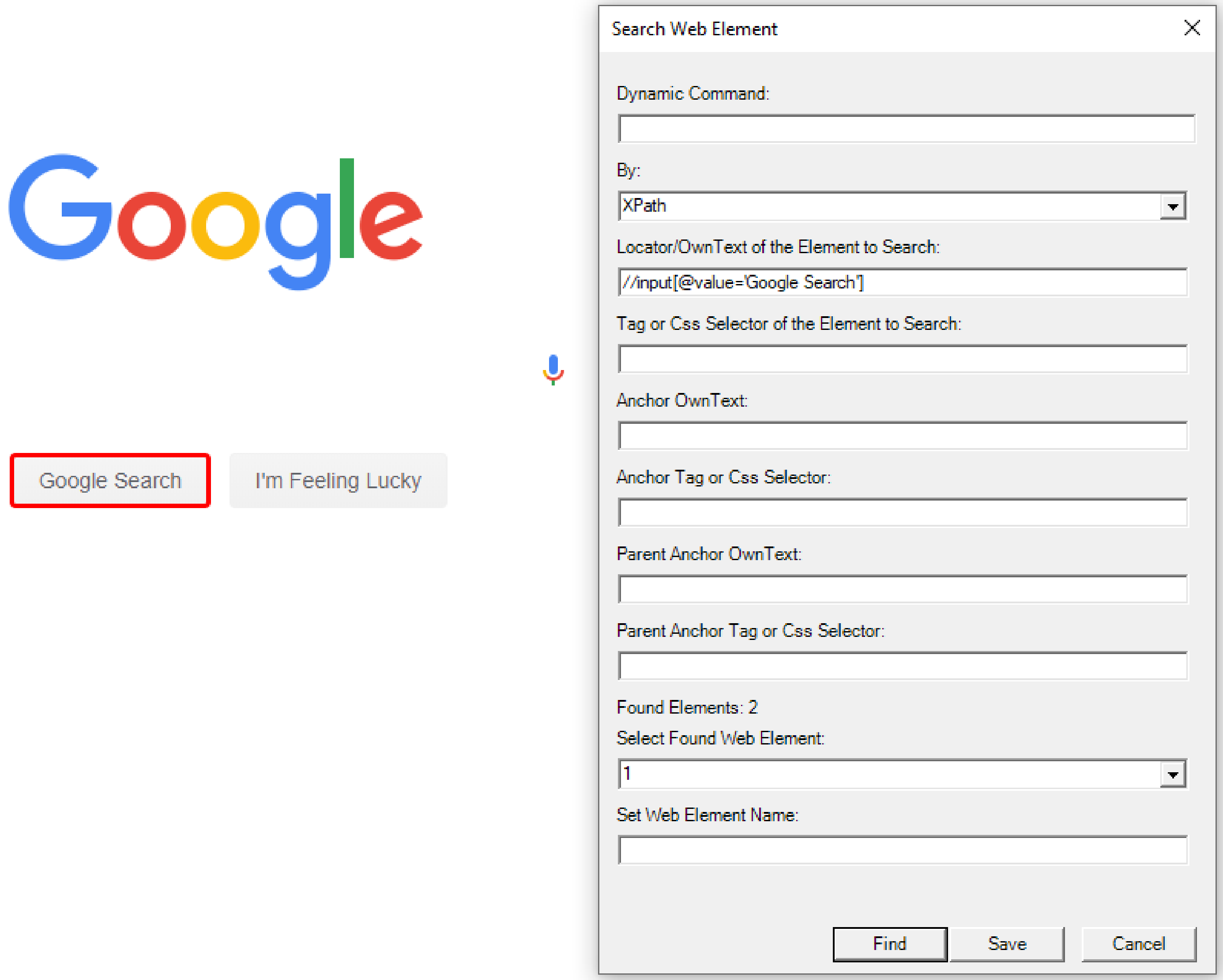
Click **Yes** or **No** and Okapi will ask further



Record Web Elements

Do you want to manually find more web elements?

If you want to manual record further, click **Yes**



Search Web Element

Dynamic Command:

By: XPath

Locator/OwnText of the Element to Search: //input[@value='Google Search']

Tag or Css Selector of the Element to Search:

Anchor OwnText:

Anchor Tag or Css Selector:

Parent Anchor OwnText:

Parent Anchor Tag or Css Selector:

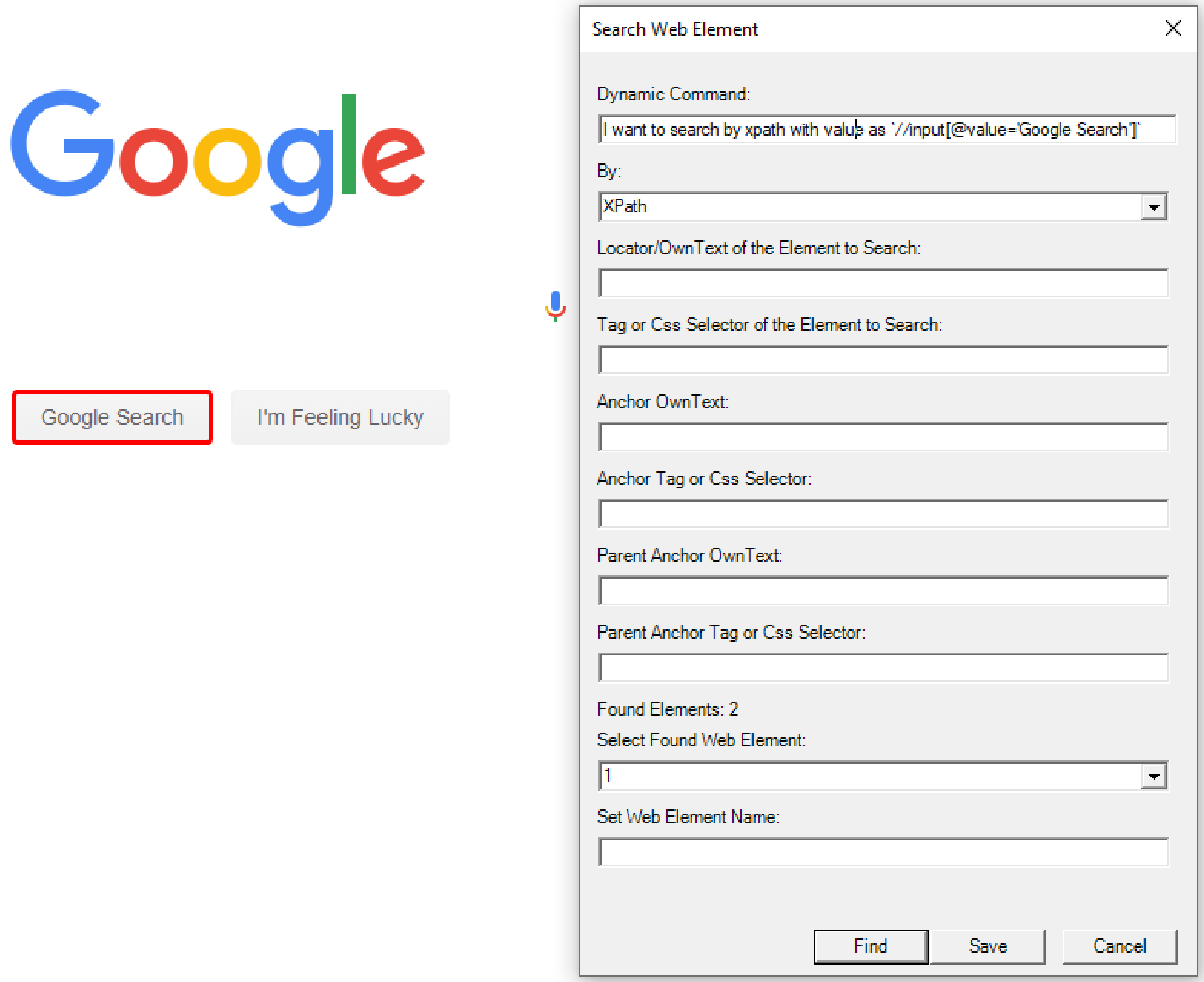
Found Elements: 2

Select Found Web Element: 1

Set Web Element Name:

You can search by xpath, css selector, class name, anchor, two anchors, etc. by input the searching data into the correct fields and click **Find**

If more than more one web object found, you need to select the index of the object you want to display from **Select Found Web Element** and click **Find** again. Okapi will highlight the right one. You can set a name for the highlighted object under **Set Web Element Name** and click **Save** to save



Search Web Element

Dynamic Command:

```
I want to search by xpath with value as `//input[@value='Google Search']`
```

By: XPath

Locator/OwnText of the Element to Search:

Tag or Css Selector of the Element to Search:

Anchor OwnText:

Anchor Tag or Css Selector:

Parent Anchor OwnText:

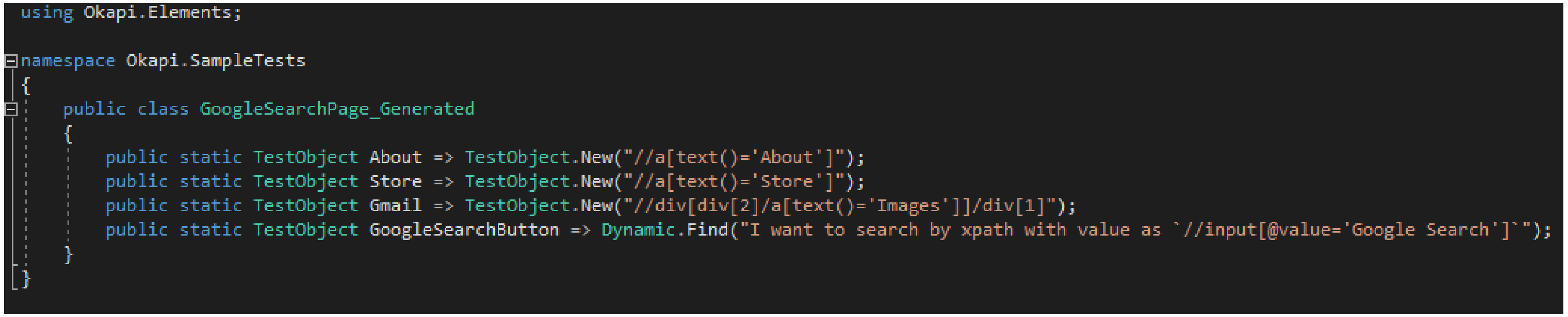
Parent Anchor Tag or Css Selector:

Found Elements: 2

Select Found Web Element: 1

Set Web Element Name:

You can search by free text as well. Any value within the free text has to be between 2 backticks (`). Any html tag has to be between < and >



```
using Okapi.Elements;

namespace Okapi.SampleTests
{
    public class GoogleSearchPage_Generated
    {
        public static TestObject About => TestObject.New("//a[text()='About']");
        public static TestObject Store => TestObject.New("//a[text()='Store']");
        public static TestObject Gmail => TestObject.New("//div[div[2]/a[text()='Images']]/div[1]");
        public static TestObject GoogleSearchButton => Dynamic.Find("I want to search by xpath with value as `//input[@value='Google Search']`");
    }
}
```

A POM class file will be saved into your specified folder, ready to be used in tests.