SPECFLOW HYBRID FRAMEWORK DOCUMEMTATION

V1.0

ZEESHAN AHMED AND HAIDER ALI KHAN

Contents

SPECFLOW HYBRID FRAMEWORK DOCUMENTATION

1. Framework Introduction

This framework is designed for testing purposes so we can test our web application and android application easily. Mainly this framework is suitable for dot net application. We can design this framework dynamically so if you can use this for android testing as well.

2. Specflow Introduction

SpecFlow is a test automation solution for .NET built upon the BDD paradigm. Use SpecFlow to define, manage and automatically execute human-readable acceptance tests in .NET projects (Full Framework and .NET Core).

SpecFlow tests are written using Gherkin, which allows you to write test cases using natural languages. SpecFlow uses the official Gherkin parser, which supports over 70 languages. These tests are then tied to your application code using so-called bindings, allowing you to execute the tests using the testing framework of your choice. You can also execute your tests using SpecFlow's dedicated test runner, SpecFlow+ Runner.

2.1 Why Specflow

- **Step definitions** - Step definitions provide the connection between Gherkin feature specifications and application interfaces for test automation.

- **Navigation to Step definitions** - Don't waste your time searching for the correct definition across your binding classes, just right-click and jump to the relevant code.

- **Hooks** - Hooks (event bindings) can be used to perform additional automation logic at specific times, such as any setup required before executing a scenario.

- **Context Injection** - SpecFlow supports a dependency injection framework that can instantiate and inject context for scenarios. This allows you to group the shared state in context classes, and inject them into every binding class that needs access to that shared state.

2.2   Specflow plus Living Doc

SpecFlow+LivingDoc is a set of tools that allows you to share and collaborate on Gherkin Feature Files with stakeholders who may not be familiar with developer tools.

- **SpecFlow+LivingDoc Generator**: If you want to generate a self-hosted HTML documentation with no external dependencies so you have the freedom to share it as you wish, then we suggest the SpecFlow plugin and command-line tool.

2.3   Specflow living doc Plugin

A plugin for SpecFlow to generate a shareable HTML Gherkin feature execution report (living documentation). Use together with SpecFlow.Plus.LivingDoc.CLI.

Note: In vscode search "Specflow living doc" and download then build your project all installed packages are shown in Dependencies -> packages and Microsoft visual studio right click on the project and go to Manage Nuget Packages and search the same string mentioned above and click on install button.

3. Specflow Framework Packages

- **Appium.Webdriver**: Selenium Webdriver extension for Appium.
- **ExcelDataReader**: Lightweight and fast library written in C# for reading Microsoft Excel files.
- **ExcelDataReader.DataSet**: ExcelDataReader extension for reading Microsoft Excel files into System.Data.DataSet.
- **Gerkin's**: Cross-platform parser for the Gherkin language, used by Cucumber, SpecFlow, and other Cucumber-based tools to parse feature files.
- **NUnit**: NUnit features a fluent assert syntax, parameterized, generic, and theory tests and is user-extensible.This package includes the NUnit 3 framework assembly, which is referenced by your tests. You will need to install version 3 of the nunit3-console program or a third-party runner that supports NUnit 3 to execute tests. Runners intended for use with NUnit 2.x will not run NUnit 3 tests correctly.
- **Selenium.Support:** Selenium is a set of different software tools each with a different approach to supporting browser automation. These tools are highly flexible, allowing many options for locating and manipulating elements within a browser, and one of its key features is the support for automating multiple browser platforms. This package contains .NET support utilities and classes that users may find useful in using Selenium WebDriver. These support classes are mainly intended to spark ideas of what is possible with Selenium WebDriver, and may not be entirely appropriate for production use**.**
- **Selenium.Webdriver.ChromeDriver:** Install Chrome Driver (Win32, macOS, and Linux64) for Selenium WebDriver into your Unit Test Project, "chromedriver(.exe)" is copied to the bin folder from the package folder when the build process NuGet package restoring ready, and no need to commit "chromedriver(.exe)" binary into source code control repository. (This package automatically downloads the latest version of chrome driver so we don't need to use webdriver manager).
- **WebDriverManager**: Automatic solved Selenium WebDriver binaries management for .NET.

Note: Webdriver manager is not working well with specflow so we recommend using "**Selenium.Webdriver.ChromeDriver**" same packages are available for different browsers as well.

Note: Some packages are available on specflow doc but these are not updated ones so when you try to use these packages you are facing some issues the most common package is "Specflow+Excel" this package is used to read data from excel but it only supports specflow 2.4 or earlier version.

4. Pre-Requisites and Packages used for the project

- .Net Developer Pack 3
- Microsoft Visual Studio / VSCode
- Go to the Extension tab and download the Specflow plugin.
- **JAVA_HOME** and **ANDROID_HOME** Path variable set
- **Path** variables set for **sdk/bin** and **sdk/platform-tools**
- Following are the Packages with their compatible versions:
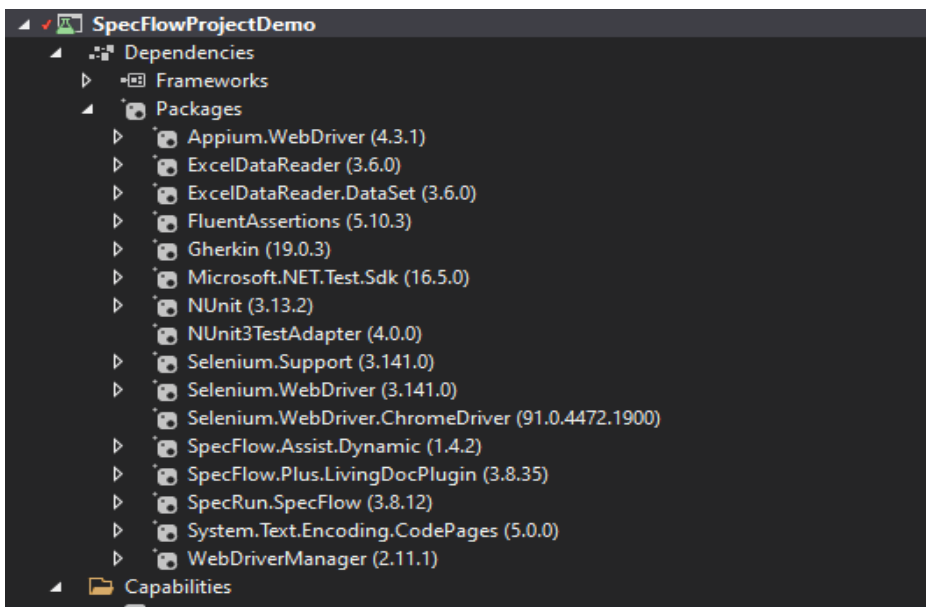
5. Project Structure

Page object model design pattern is used in this framework and the complete structure of this framework is shown below.
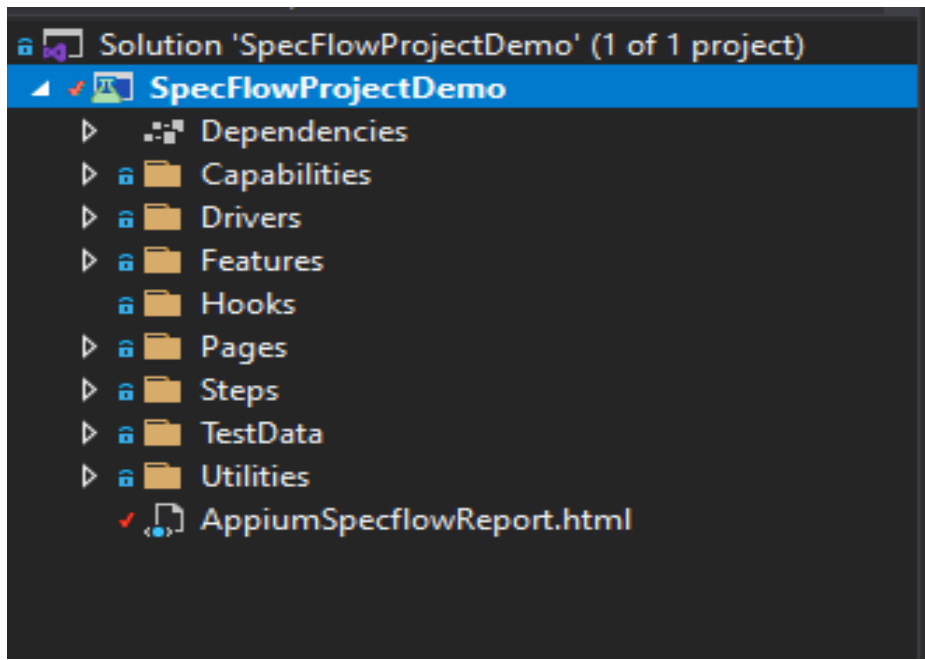
6. Page Factory In C#

- Why **not** to Implement **Page Factory** In **C#**

Here is the post where Jim Evans (the maintainer of C# Selenium binding) explains the issue you mentioned. In short:

3

> Its only now that I have started to write Selenium tests in C# that I understand why the implementation was described as deeply flawed. The reason for dropping support now is due to it using proxies that have been deprecated from dotnet, but the reality is that they probably should never have existed in the first place. Let me explain...
>
> When I was trying to get my defined rootElement PageBlocks working in Java I had problems at first with trying to instantiate the Page Objects when the Page was not displayed in the browser. You need to have lazy instantiation and as far as I am aware, you need the proxies as used in the Java PageFactory in order to do this. This is not however true in C# as properties allow you to define a call that is not made on instantiation. This is explained very clearly by Jim Evans in the issue where the deprecation of the .net proxies was raised. Whilst Liraz Shay was correct that this is more verbose, I totally understand that using complex proxies and reflection is an unjustifiable maintenance problem (and probably slower performing too.)

There you will also find the approach to use without `PageFactory` class.

Share Improve this answer Follow

answered Jan 30 '19 at 16:08

Alexey R.
11.2k ● 4 ● 15 ● 36

7. Project Details

- **Dependencies**: All the dependencies and packages are available in this folder
- **Driver**: Driver class is defined as a hook for **closing** all the **running drivers** including **appium services and browser drivers.**
- **Features**: All feature files define in this folder (when you want to create a feature file then you can directly change the file extension or you can create the file as a specflow feature file).
- **Hooks**: If you need any hooks then define them in this folder.
- **Pages**: Web page class define in this folder
- **Steps**: Steps definitions of a feature defined in this folder (when you generate steps definitions file you need set path to steps folder)
- **TestResult**: Test report is generated in this folder
- **Utilities**: All the utilities file files define in this folder like JSON reader excel reader and Capabilities.cs.

8. VSCode Setup

- For Easy scripting download and enable the following extensions for your specflow project:
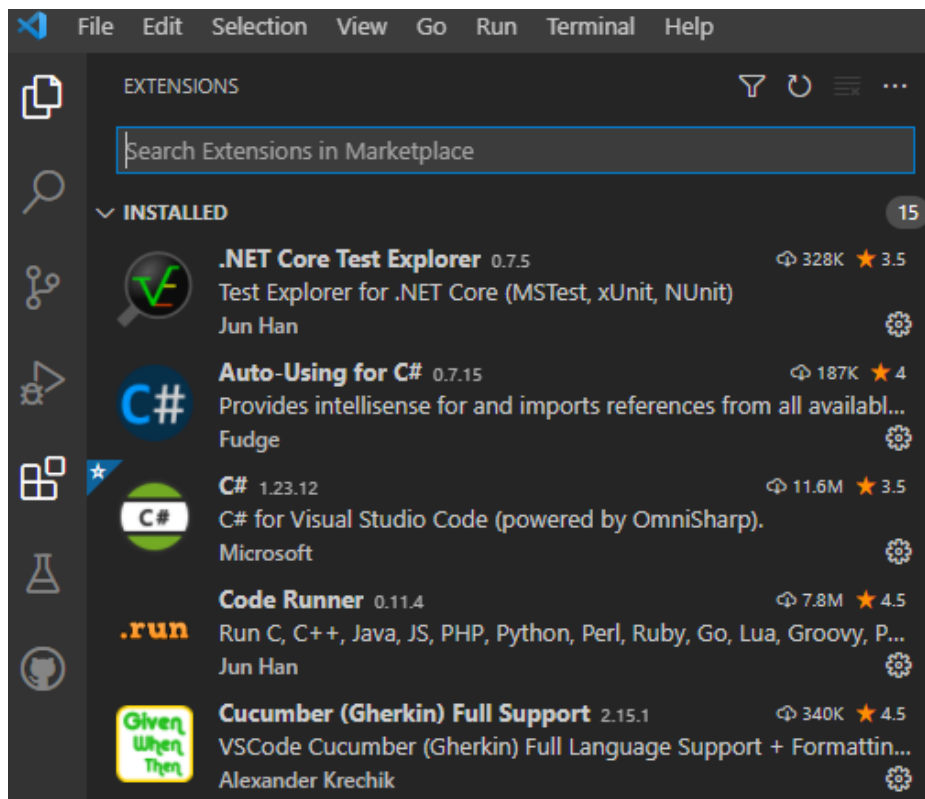
- VSCode tasks have been created. To run the tasks you have to do the following:
  - Press **Ctrl+Shit+P** to open the **runner** and type "Task" and choose the required task.
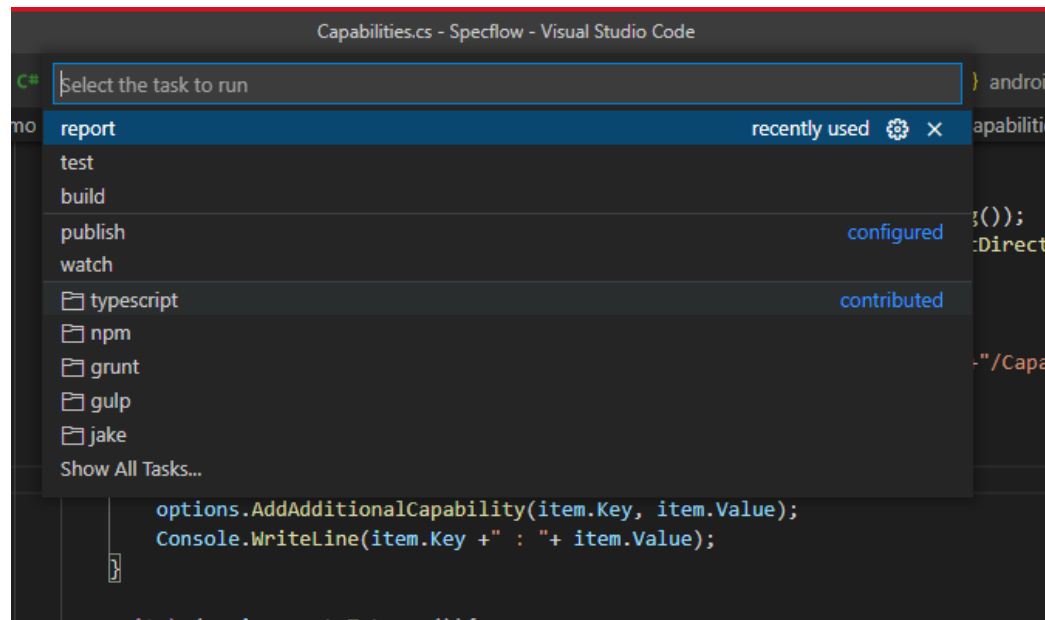
  - **Build Task** – Builds your code for your latest changes
  - **Test Task** – Runs all your tests defined in the feature folder (E2E)
  - **Report Task** – Generates a report from the last executed test cases.
- If you want to run the tasks manually from any Powershell or terminal, you can do the following:
  - Open the project in the cmd:
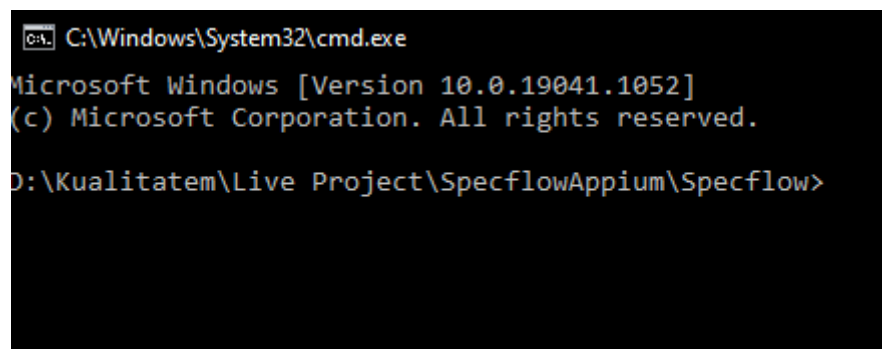
  - You can use the following commands:
    - **Build** – dotnet build
    - **Test** – dotnet test or dotnet test --filter TestCategory=mytag
    - **Report**- livingdoc test-assembly
      bin\Debug\netcoreapp3.1\SpecFlowProjectDemo.dll -t
      bin\Debug\netcoreapp3.1\TestExecution.json

9. Hybrid Driver Instantiation

- **Capabilities.cs** is defined as a **Static Class** so you just need to call the class directly and you will have access to the following methods and objects:
  - Object driver
  - setup (method)
  - stopAppium(method)
  - stopBrowserDriver(method)
- To **instantiate** you need to use the **setup method and pass a variable** that should denote the **environment** you want to work on. Following are the environments that have been set:
  - Android
  - iOS
  - Windows
  - Browser
- To instantiate a specific **browser** you need to write which type of browser you want to invoke in Capabilities -> browserConfig.json. Following are the **supported browsers**:
  - Chrome
  - Firefox
  - Opera
  - Edge
- In this framework, we have **implemented** the **driver instantiation** at **feature file level**. So you just need to write a single feature line and pass the application as a variable:
  - Given the application is open

    or
  - When the application is open

```
22    @test
23  v Scenario: Testing Mobile and Browser Config
24  v     Given the application is open
25              | application ||      You, 18 hours ago • Specflow
26              | android   |
27        Then user clicks on continue as guest
28        Then user clicks on coronavirus
29        Then user clicks on material
30  v     Given the application is open
31              | application |
32              | browser      |
33        Given open browser
34  v     When user enter username  and password
35              | Username | Password   |
36              | Perfecto | Password2! |
37        Then click on sign In button
38        Then verify user login successfully
```