Chapter 37: BDD Unit Testing in Xamarin.Forms

Section 37.1: Simple Specflow to test commands and navigation with NUnit Test Runner

Why do we need this?

The current way to do unit testing in Xamarin.Forms is via a platform runner, so your test will have to run within an ios, android, windows or mac UI environment : Running Tests in the IDE

Xamarin also provides awesome UI testing with the Xamarin.TestCloud offering, but when wanting to implement BDD dev practices, and have the ability to test ViewModels and Commands, while running cheaply on a unit test runners locally or on a build server, there is not built in way.

I developed a library that allows to use Specflow with Xamarin.Forms to easily implement your features from your Scenarios definitions up to the ViewModel, independently of any MVVM framework used for the App (such as XLabs, MVVMCross, Prism)

If you are new to BDD, check Specflow out.

Usage:

- If you don't have it yet, install the specflow visual studio extension from here (or from you visual studio IDE): https://visualstudiogallery.msdn.microsoft.com/c74211e7-cb6e-4dfa-855d-df0ad4a37dd6
- Add a Class library to your Xamarin.Forms project. That's your test project.
- Add SpecFlow.Xamarin.Forms package from nuget to your test projects.
- Add a class to you test project that inherits 'TestApp', and register your views/viewmodels pairs as well as adding any DI registration, as per below:

```
public class DemoAppTest : TestApp
{
    protected override void SetViewModelMapping()
    {
        TestViewFactory.EnableCache = false;

        // register your views / viewmodels below
        RegisterView<MainPage, MainViewModel>();
    }

    protected override void InitialiseContainer()
    {
        // add any di registration here
        // Resolver.Instance.Register<TInterface, TType>();
        base.InitialiseContainer();
    }
}
```

- Add a SetupHook class to your test project, in order to add you Specflow hooks. You will need to bootstrap the test application as per below, providing the class you created above, and the your app initial viewmodel:

```
[Binding]
public class SetupHooks : TestSetupHooks
```

```
    {
        /// <summary>
        ///     The before scenario.
        /// </summary>
        [BeforeScenario]
        public void BeforeScenario()
        {
            // bootstrap test app with your test app and your starting viewmodel
            new TestAppBootstrap().RunApplication<DemoAppTest, MainViewModel>();
        }
    }
```

- You will need to add a catch block to your xamarin.forms views codebehind in order to ignore xamarin.forms framework forcing you to run the app ui (something we don't want to do):

```
        public YourView()
        {
            try
            {
                InitializeComponent();
            }
            catch (InvalidOperationException soe)
            {
                if (!soe.Message.Contains("MUST"))
                    throw;
            }
        }
```

- Add a specflow feature to your project (using the vs specflow templates shipped with the vs specflow extension)
- Create/Generate a step class that inherits TestStepBase, passing the scenarioContext parameter to the base.
- Use the navigation services and helpers to navigate, execute commands, and test your view models:

```
  [Binding]
    public class GeneralSteps : TestStepBase
    {
        public GeneralSteps(ScenarioContext scenarioContext)
            : base(scenarioContext)
        {
            // you need to instantiate your steps by passing the scenarioContext to the base
        }

        [Given(@"I am on the main view")]
        public void GivenIAmOnTheMainView()
        {
            Resolver.Instance.Resolve<INavigationService>().PushAsync<MainViewModel>();

Resolver.Instance.Resolve<INavigationService>().CurrentViewModelType.ShouldEqualType<MainViewModel>
();
        }

        [When(@"I click on the button")]
        public void WhenIClickOnTheButton()
        {
            GetCurrentViewModel<MainViewModel>().GetTextCommand.Execute(null);
        }

        [Then(@"I can see a Label with text ""(.*)""")]
        public void ThenICanSeeALabelWithText(string text)
```

```
        {
            GetCurrentViewModel<MainViewModel>().Text.ShouldEqual(text);
        }
    }
```

Section 37.2: Advanced Usage for MVVM

To add to the first example, in order to test navigation statements that occurs within the application, we need to provide the ViewModel with a hook to the Navigation. To achieve this:

- Add the package SpecFlow.Xamarin.Forms.IViewModel from nuget to your PCL Xamarin.Forms project
- Implement the IViewModel interface in your ViewModel. This will simply expose the Xamarin.Forms INavigation property:
- `public class MainViewModel : INotifyPropertyChanged, IViewModel.IViewModel { public INavigation Navigation { get; set; }`
- The test framework will pick that up and manage internal navigation
- You can use any MVVM frameworks for you application (such as XLabs, MVVMCross, Prism to name a few. As long as the IViewModel interface is implemented in your ViewModel, the framework will pick it up.