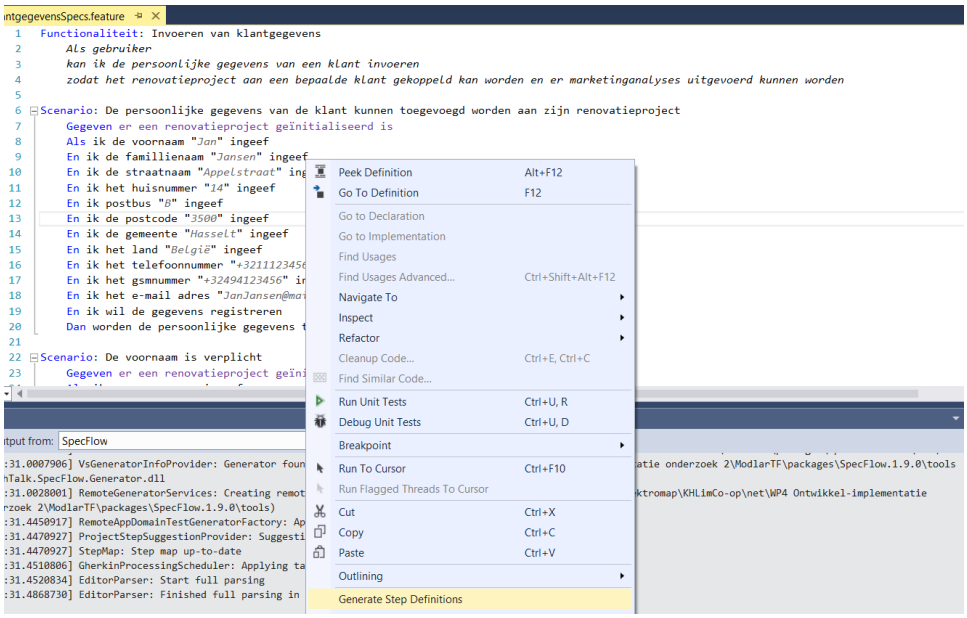
How To's

Abbreviations

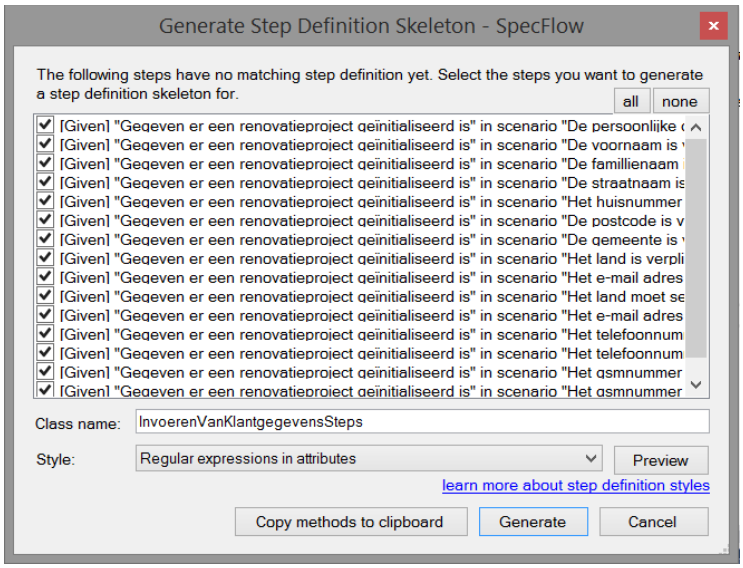
- RMC: Right mouse click
- Spec: Specification

Generate step definitions

1) RMC on your specs
2) Select generate step definitions

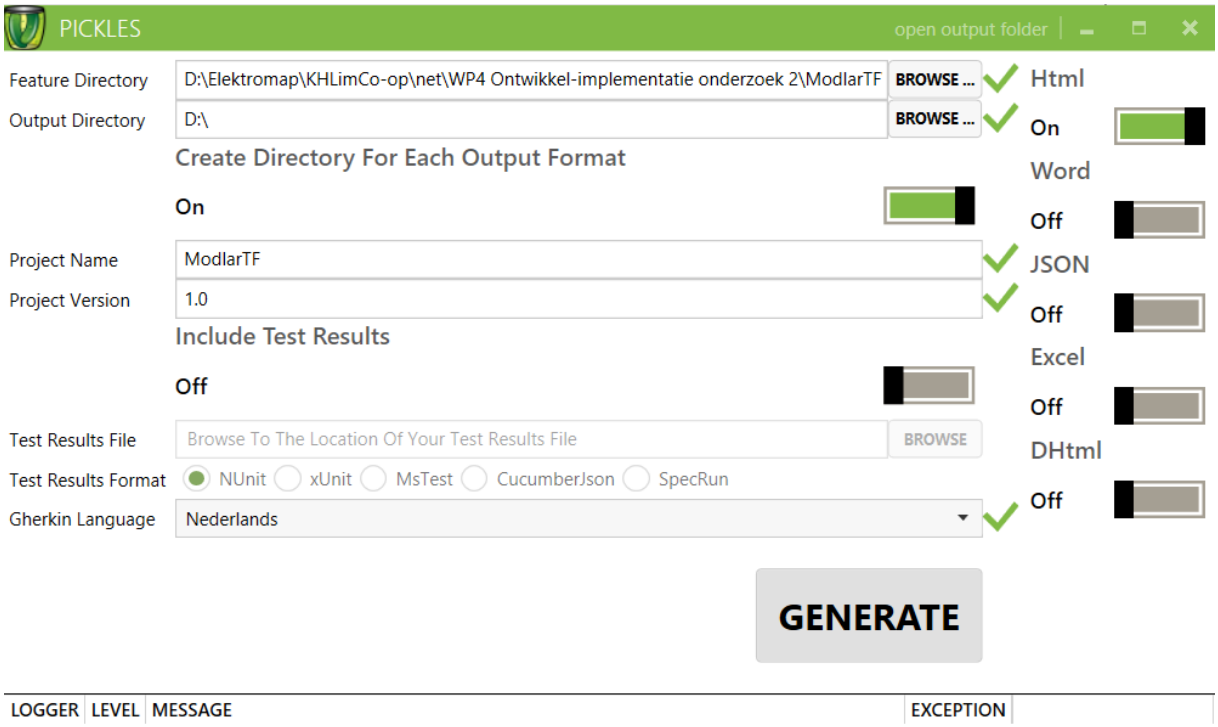


3) Choose to save them to a file or copy them to the clipboard



Create a report with specifications before step generation

1) Write your specs
2) Download and install Pickles[1][2]
3) Run pickles from the install folder
4) Enter the feature directory
5) Enter the output directory
6) Enter project name
7) Enter project version
8) Enter Gherkin language
9) Provide miscellaneous settings(output format,...)



10) Press generate

[1] http://docs.picklesdoc.com/en/latest/GettingStarted/

[2] In this “how to”, the choise is made to use a zip-package and the GUI version

Insert tables in steps

1) Use the pipe symbol to create a table(placed beneath the step which requires the table)

```
Scenario: De persoonlijke gegevens van de klant kunnen toegevoegd worden
    Gegeven een customer met geldige data
    | Veld           | Waarde              |
    | Voornaam       | Jan                 |
    | Famillienaam   | Jansen              |
    | Straatnaam     | Appelstraat         |
    | Huisnummer     | 14                  |
    | Postbus        | B                   |
    | Postcode       | 3500                |
    | Gemeente       | Hasselt             |
    | Land           | België              |
    | Telefoonnummer | +3211123456         |
    | Gsmnummer      | +32494123456        |
    | E-mail adres   | JanJansen@mail.com  |
    Als ik de gegevens wil valideren
    Dan wordt de customer als geldig aanschouwd
```

2) Generate step definitions
   The step header should look like displayed in the image below

```
[Given(@"een customer met geldige data")]
0 references
public void GegevenEenCustomerMetGeldigeData(Table data)
{
```

Introduce variables in steps

1) Write a common step
2) Generate the step definition
3) Place (.*) in every position into the method attribute of the step to identify the position where data is considered to be a variable

```
[Then(@"krijg ik een errormelding ""(.*)"" terug")]
```

4) Add the parameters between the brackets of the method. There should be an equal number of parameters and positions marked with (.*) in the method attribute(step)

```
[Then(@"krijg ik een errormelding ""(.*)"" terug")]
0 references
public void DanKrijgIkEenErrormeldingTerug(String error)
{
```

5) Copy and paste the common step to all location that use that step, changing only the variable part.

Change the language for writing specifications

1) Open the app.config
2) After the opening specflow tag, insert following tag

```
<language feature="language" />
```

Where language equals the required language[3]

Mark steps as pending

1) Navigate to the step that should be pending(RMC on the targeted step -> Go To Step Definition)
2) Write "`ScenarioContext.Current.Pending();`" in the step method

Create a cyclic(data driven) test

1) Create a scenario outline(used as a scaffold for the test)

```
Abstract Scenario: Het e-mail adres moet in het juiste format staan
```

2) Write the necessary steps and mark the input variables between < and >

```
Als ik een e-mail adres ingeef dat niet het correcte format heeft: <Fout_mailadres>
```

3) Write the examples(data for each run) beneath the scenario outline in table format

```
Abstract Scenario: Het e-mail adres moet in het juiste format staan
    Gegeven er een renovatieproject geïnitialiseerd is
    Als ik een e-mail adres ingeef dat niet het correcte format heeft: <Fout_mailadres>
    En ik alle andere parameters correct ingeef
    En ik wil de gegevens registreren
    Dan krijg ik een errormelding "Het e-mail adres moet in het format gebruiker@provider.domein staan" terug
    Voorbeelden:
    | Fout_mailadres     |
    | JanJansen          |
    | JanJansen@         |
    | JanJansen@mail     |
    | JanJansen%mail.com |
```

4) After generating the step definitions, make sure the varaibles are correctly recognised(see "Introduce variables in steps" for more info)

[3] Possible values for the feature language are found on following site: https://github.com/cucumber/gherkin/blob/master/lib/gherkin/i18n.json

Run a test suite

1) RMC on the feature that needs to be run
2) Click on “Run specflow scenarios” on the pop up window

Run a single test

1) RMC on the scenario that needs to be run
2) Click on “Run specflow scenarios” on the pop up window [4]

Connect to MS-Coded UI

1) Add an Coded UI test project to your solution
2) Install the Specflow nuget
3) Add a new class library to your solution named: CodedUIGeneratorProvider
4) Install Specflow and Specflow.Customplugin nuget
5) Add a class named CodedUIGeneratorProvider to the class library
6) Paste in the code you find in the attachment A
7) Rename the assembly name and namespace via properties to: CodedUIGeneratorProvider.Generator.SpecFlowPlugin
8) Build the project
9) Go back to the test project
10) Set the unit test provider to MS test(make sure no other test provider is referenced)

```
<unitTestProvider name="MsTest" />
```

11) Add following plugin reference to the app.config

```
<plugins>
  <add name="CodedUIGeneratorProvider" path="path to plugin" type="Generator" />
</plugins>
```

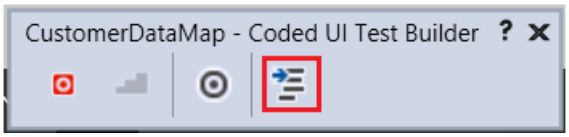
Where “path to plugin” is the full path to the DLL of the newly created CodedUIGeneratorProvider

12) Add a folder named specs to the project
13) Add your specflow scenario’s(+ steps) to the specs folder
14) Add a folder named Map to the project
15) Add a Coded UI test map to the folder(one per view is advised) and give it a logic name(e.g. CustomerInfoMap)
16) RMC on the test map -> Click on with Coded UI Test Builder
17) A window like the image below should appear in the lower right corner of the screen

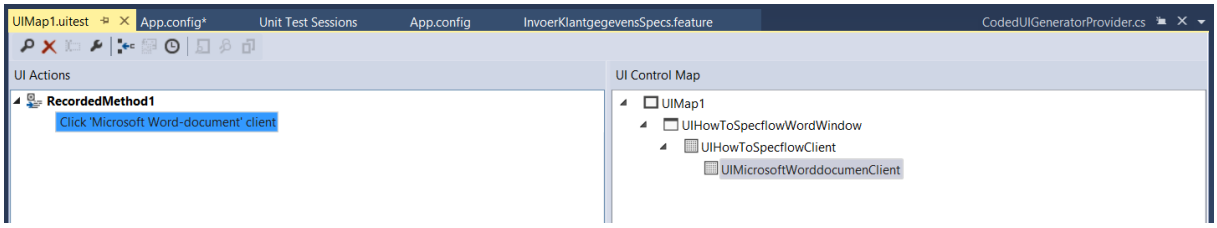
[4] You can’t single out a test from a scenario outline

18) Start up your application
19) Click on the record button

20) Perform all actions(clicking ,modifying tekst,...) that need to be done to perform your test. Make sure you have at least clicked on all items related to the test, otherwise the element can't be found when connecting to specflow
21) Click on the record button again
22) Click on the generate test code button



23) Click on Test and Generate
24) Close the Coded UI Test Builder
25) Double click on the test map. A window should pop up, that looks like the image below



26) RMC on the record method(left pane) -> Click on move code to ...
27) Press OK to confirm

28) Open the .cs file of your UI map. It should be written in the same style as the image below

```
public void RecordedMethod1()
{
    #region Variable Declarations
    WpfComboBox uICmbLanguagesComboBox = this.UIRemfr0Window.UIREMFR0componentviewsPane.UICmbLanguagesComboBox;
    WpfButton uINieuwprojectButton = this.UIRemfr0Window.UIREMFR0componentviewsPane.UINieuwprojectButton;
    #endregion

    // Select 'Nederlands' in 'cmbLanguages' combo box
    uICmbLanguagesComboBox.SelectedItem = this.RecordedMethod1Params.UICmbLanguagesComboBoxSelectedItem;

    // Click 'Nieuw project' button
    Mouse.Click(uINieuwprojectButton, new Point(83, 14));
}

1 reference
public virtual RecordedMethod1Params RecordedMethod1Params
{
    get
    {
        if ((this.mRecordedMethod1Params == null))
        {
            this.mRecordedMethod1Params = new RecordedMethod1Params();
        }
        return this.mRecordedMethod1Params;
    }
}

private RecordedMethod1Params mRecordedMethod1Params;
}
/// <summary>
/// Parameters to be passed into 'RecordedMethod1'
/// </summary>
[GeneratedCode("Coded UITest Builder", "12.0.31101.0")]
3 references
public class RecordedMethod1Params
```

29) Look for the mappings of your element in this file.

```
WpfComboBox uICmbLanguagesComboBox = this.UIRemfr0Window.UIREMFR0componentviewsPane.UICmbLanguagesComboBox;
WpfButton uINieuwprojectButton = this.UIRemfr0Window.UIREMFR0componentviewsPane.UINieuwprojectButton;
#endregion

// Select 'Nederlands' in 'cmbLanguages' combo box
uICmbLanguagesComboBox.SelectedItem = this.RecordedMethod1Params.UICmbLanguagesComboBoxSelectedItem;

// Click 'Nieuw project' button
Mouse.Click(uINieuwprojectButton, new Point(83, 14));
```

30) Rewrite the file in such a way that it contains methods that resemble the steps in your scenario's. And pass the value to the UI elements via the mappings that were originally generated. This should give you a result that looks like the image below

```
1 reference
public void enterFirstName(String value)
{
    WpfEdit uITxtFirstnameEdit = UIRemfr0Window.UIREMFR0componentviewsPane.UITxtFirstnameEdit;
    uITxtFirstnameEdit.Text = value;
}

1 reference
public void enterLastName(String value)
{
    WpfEdit uITxtLastnameEdit = UIRemfr0Window.UIREMFR0componentviewsPane.UITxtLastnameEdit;
    uITxtLastnameEdit.Text = value;
}
```

31) Repeat steps 15 -> 30 for every view/feature

32) Create a wrapper class for all the UI maps

```
18 references
public static class ModlarUIMapAbstract
{
    private static CustomerDataMap _customerDataUIMap;
    private static IntroMap _introUIMap;
    15 references
    public static CustomerDataMap customerDataUIMap
    {
        get { return _customerDataUIMap ?? (_customerDataUIMap = new  CustomerDataMap()); }
    }

    3 references
    public static IntroMap introMap
    {
        get { return _introUIMap ?? (_introUIMap = new IntroMap()); }
    }
}
```

33) Go back to your specs

34) Generate the step definitions

35) Wire the steps to the UI map

```
[When(@"ik wil de gegevens registreren")]
0 references
public void AlsIkWilDeGegevensRegistreren_UI()
{
    ModlarUIMapAbstract.customerDataUIMap.enterFirstName(_voornaam);
    ModlarUIMapAbstract.customerDataUIMap.enterLastName(_famillienaam);
```

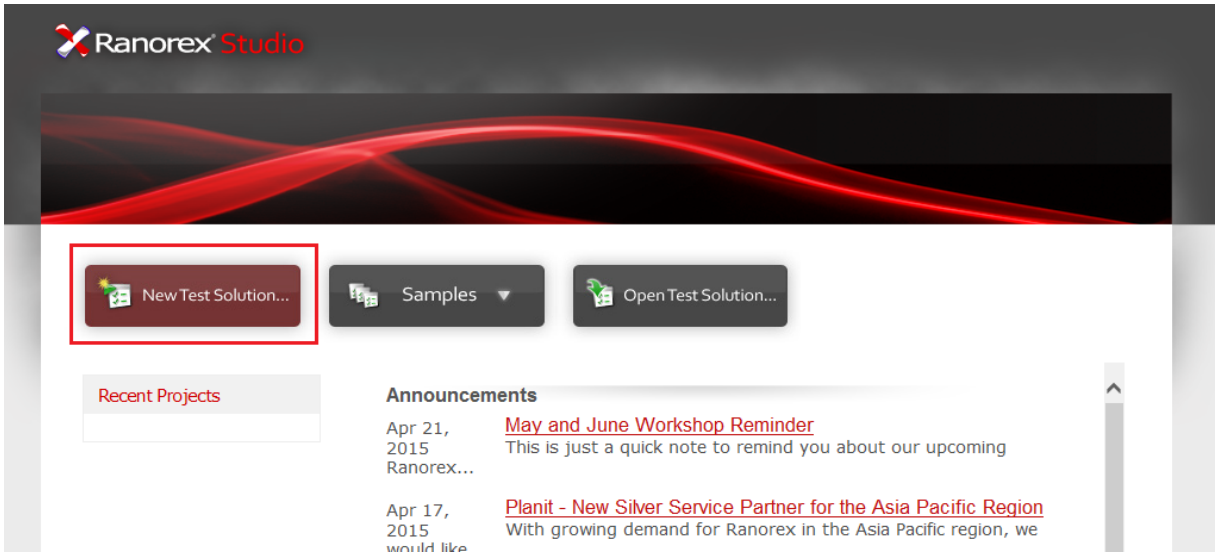
36) Write a setup step that starts the application, and a teardown step that closes the application

```
[AfterScenario]
0 references
public void tearDown()
{
    _pr.Close();
}
```

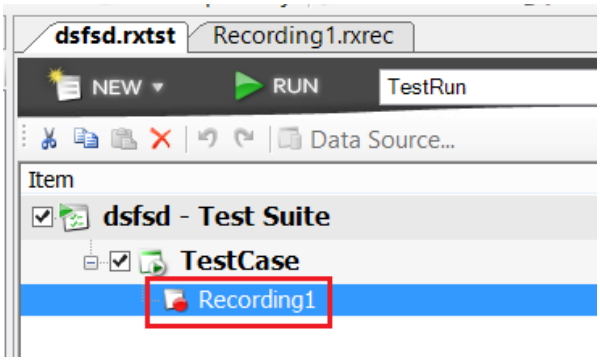
```
[BeforeScenario]
0 references
public void resetParameters()
{
    _pr = Process.Start(@"D:\Elektromap\KHLim\Projecten\Modlar\REMFR0\REMFR0\bin\Debug\REMFR0.exe");
}
```

Connect to Ranorex

1) Install Ranorex
2) Create a new test project
3) Install the SpecFlow nuget
4) Add Ranorex.Core to the reference list
5) Create your specifications
6) Generate your stepdefinitions
7) Start Ranorex
8) Generate a new test solution



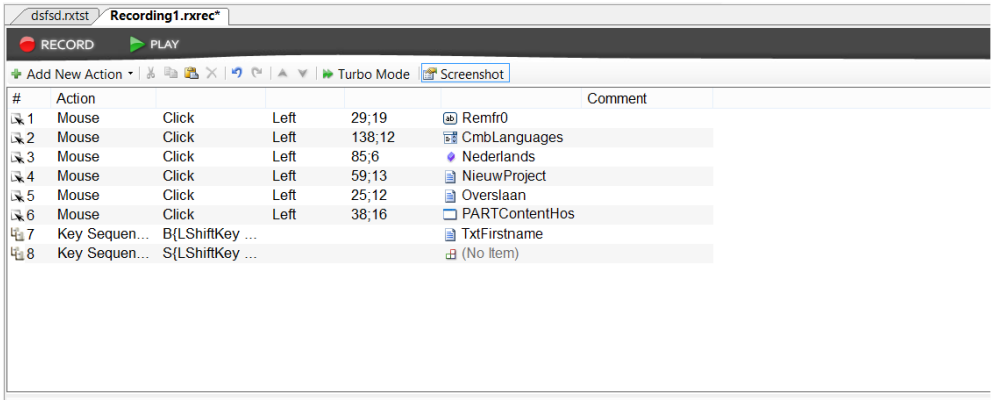
9) Double click on the recording



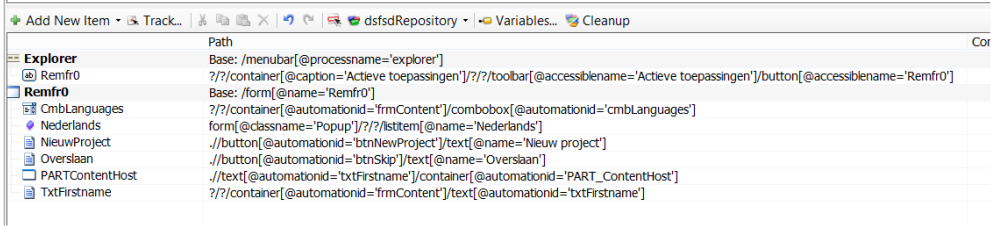
10) Start your application
11) Press record in Ranorex
12) Set the record settings to “Global recording”
13) Perform all actions(clicking ,modifying tekst,...) that need to be done to perform your test. Make sure you have at least clicked on all items related to the test, otherwise the element can’t be found when connecting to specflow
14) Press stop recording
15) You go back to the ranorex main window

16) Now you see all the performed actions and the UI path to the controls

Actions ⇨



Path's ⇨



17) Go back to your steps

18) Via the Ranorex core, wire up the data/actions to your application using the paths you can find in Ranorex. The path to a control is made up out of 2 parts. The base path and the control path. An example is displayed in the image below.

```
[When(@"ik wil de gegevens registreren")]
0 references
public void AlsIkWilDeGegevensRegistreren_UI()
{
    Text txt = _frm.FindSingle<Text>("/form[@name='Remfr0']/?/?/text[@automationid='txtFirstname']");
    txt.TextValue = _voornaam;

    txt = _frm.FindSingle<Text>("/form[@name='Remfr0']/?/?/text[@automationid='txtLastname']");
    txt.TextValue = _familienaam;
```

Basepath

Control path

Attachments

A. Code for CodedUiGeneratorProvider

```
using CodedUIGeneratorProvider.Generator.SpecFlowPlugin;
using TechTalk.SpecFlow.Infrastructure;
[assembly:GeneratorPlugin(typeof(CodedUIGeneratorPlugin))]

namespace CodedUIGeneratorProvider.Generator.SpecFlowPlugin
{
    using System.CodeDom;

    using BoDi;

    using TechTalk.SpecFlow.Generator;
    using TechTalk.SpecFlow.Generator.Configuration;
    using TechTalk.SpecFlow.Generator.Plugins;
    using TechTalk.SpecFlow.Generator.UnitTestProvider;
    using TechTalk.SpecFlow.UnitTestProvider;
    using TechTalk.SpecFlow.Utils;

    /// <summary>
    /// The CodedUI generator plugin.
    /// </summary>
    public class CodedUIGeneratorPlugin : IGeneratorPlugin
    {
        /// <summary>
        /// The register dependencies.
        /// </summary>
        /// <param name="container">
        /// The container.
        /// </param>
        public void RegisterDependencies(ObjectContainer container)
        {
        }

        /// <summary>
        /// The register customizations.
        /// </summary>
        /// <param name="container">
        /// The container.
        /// </param>
        /// <param name="generatorConfiguration">
        /// The generator configuration.
        /// </param>
        public void RegisterCustomizations(ObjectContainer container,
SpecFlowProjectConfiguration generatorConfiguration)
        {
            container.RegisterTypeAs<CodedUIGeneratorProvider,
IUnitTestGeneratorProvider>();
            container.RegisterTypeAs<MsTest2010RuntimeProvider,
IUnitTestRuntimeProvider>();
        }

        /// <summary>
        /// The register configuration defaults.
        /// </summary>
        /// <param name="specFlowConfiguration">
        /// The spec flow configuration.
        /// </param>
```

```
        public void RegisterConfigurationDefaults(SpecFlowProjectConfiguration
specFlowConfiguration)
        {
        }
    }

    /// <summary>
    /// The CodedUI generator.
    /// </summary>
    public class CodedUIGeneratorProvider : MsTest2010GeneratorProvider
    {
        /// <summary>
        /// Initializes a new instance of the <see cref="CodedUiGeneratorProvider"/>
class.
        /// </summary>
        /// <param name="codeDomHelper">
        /// The code dom helper.
        /// </param>
        public CodedUIGeneratorProvider(CodeDomHelper codeDomHelper)
            : base(codeDomHelper)
        {
        }

        /// <summary>
        /// The set test class.
        /// </summary>
        /// <param name="generationContext">
        /// The generation context.
        /// </param>
        /// <param name="featureTitle">
        /// The feature title.
        /// </param>
        /// <param name="featureDescription">
        /// The feature description.
        /// </param>
        public override void SetTestClass(TestClassGenerationContext
generationContext, string featureTitle, string featureDescription)
        {
            base.SetTestClass(generationContext, featureTitle, featureDescription);

            foreach (CodeAttributeDeclaration declaration in
generationContext.TestClass.CustomAttributes)
            {
                if (declaration.Name ==
"Microsoft.VisualStudio.TestTools.UnitTesting.TestClassAttribute")
                {
                    generationContext.TestClass.CustomAttributes.Remove(declaration);
                    break;
                }
            }

            generationContext.TestClass.CustomAttributes.Add(new
CodeAttributeDeclaration(new
CodeTypeReference("Microsoft.VisualStudio.TestTools.UITesting.CodedUITestAttribute")))
;
        }
    }
}
```