Understanding SpecFlow Fundamentals

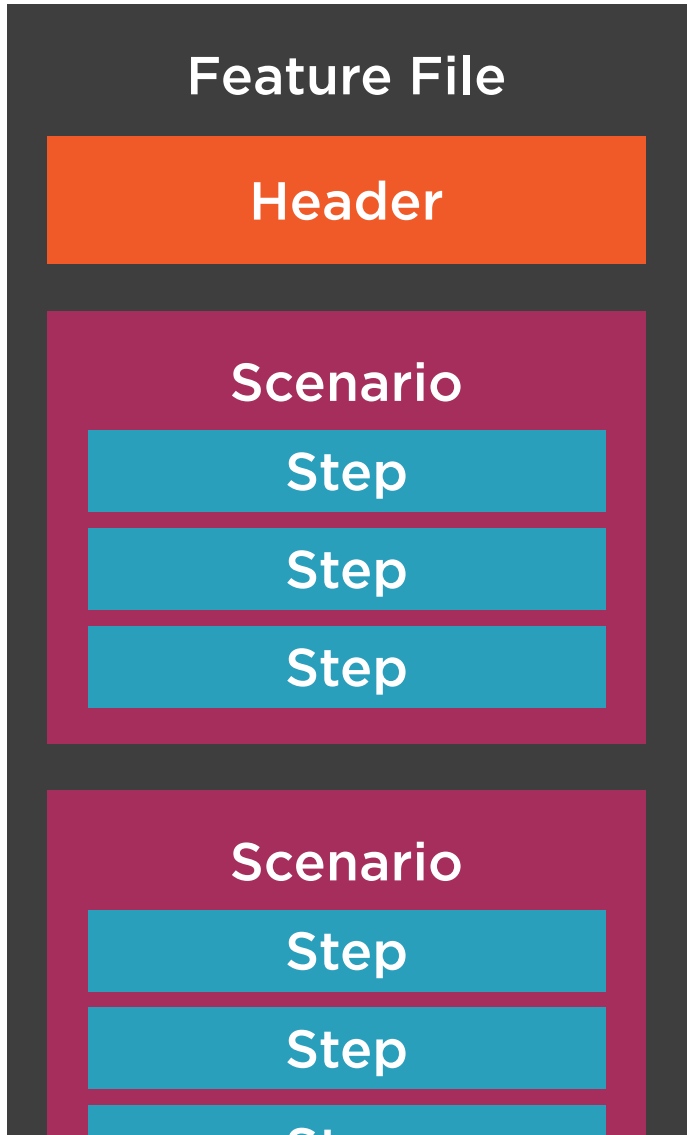
Jason Roberts
.NET MVP

@robertsjason dontcodetired.com

Overview

- Understanding feature files
- Feature headers
- Scenario steps (“given”, “when”, “then”)
- Comments and tags in feature files
- Step definition code
- Installation overview and NuGet packages
- Getting started in Visual Studio
- Step definition binding styles

Feature File Structure



Name and description of feature

Scenario name

Logical steps (high level / business oriented)

```
Feature: PlayerCharacter
    In order to play the game
    As a human player
    I want my character attributes to be correctly represented
```

Header

Feature name

Free textual description of the feature

Any format / number of lines

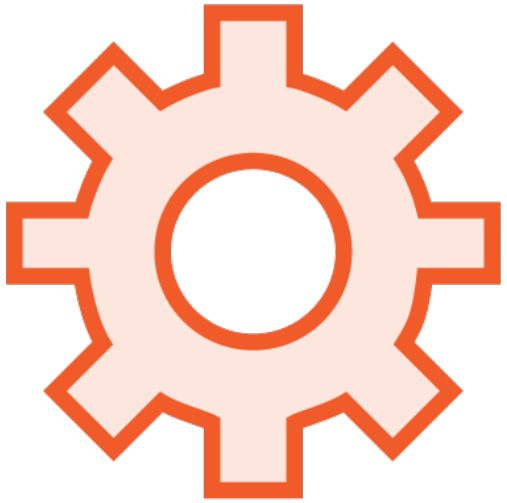
```
Feature: PlayerCharacter

    Players exist in game world and have attributes such as
    magical power and damage protection. Different character
    races have different basic attributes, for example Elves
    get an extra 20 points of damage resistance.
```

Format-less header

Writing Feature Headers

Who benefits or is interested in this feature?



What is required/necessary?

Why is this feature important/valuable?

General Guidelines

Length / brevity

- Long enough to hold enough detail, not so long as no one bothers to read it

Be specific about the "who"

- "HR manager", rather than "user"

Adjectives bring things to life

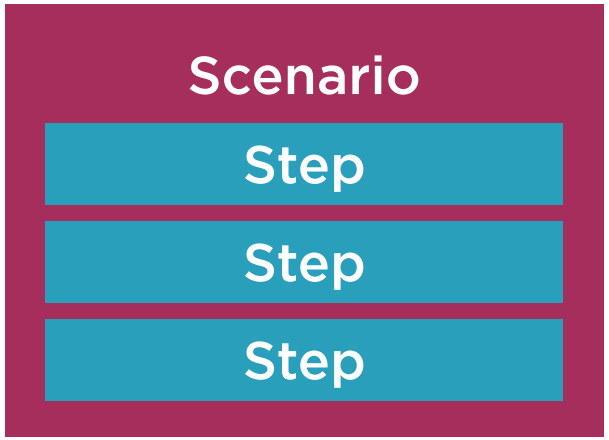
- "Busy HR manager" or "Frustrated HR manager"

Use a format (or formats) that increase ease of communication for all readers

“[Gherkin] … is a Business Readable, Domain Specific Language that lets you describe software’s behaviour without detailing how that behaviour is implemented.”

https://github.com/cucumber/cucumber/wiki/Gherkin

Gherkin Steps



Given - Setup starting state of system

When - Describe the action that takes place

Then - Observe and verify outcome / end state

```
Scenario: Starting health
    Given I'm a new player
    When I take 40 damage
    Then my health should now be 60
```

Given, When, Then

```
Scenario: Starting health
    Given I’m a new player
    Given I’m an Elf
    Given I’m not wearing armour
    When I take 40 damage
    Then my health should now be 60
    Then I should still be standing
```

And, But

```
Scenario: Starting health
    Given I'm a new player
        And I'm an Elf
        But I'm not wearing armour
    When I take 40 damage
    Then my health should now be 60
        And I should still be standing
```

And, But

Can use in Given, When, or Then phases

Improves readability / fluentness

```
# This scenario is a common one
Scenario: Starting health
    Given I'm a new player
    # All players start with 100 health
    When I take 40 damage
    Then my health should now be 60
```

Comments

Lines that start with #

```
@health @fighting
Scenario: Starting health
    Given I'm a new player
    When take 40 damage
    Then my health should now be 60
```

Tags

Additional way to organize features and scenarios

Feature or scenario level

Can have multiple tags set

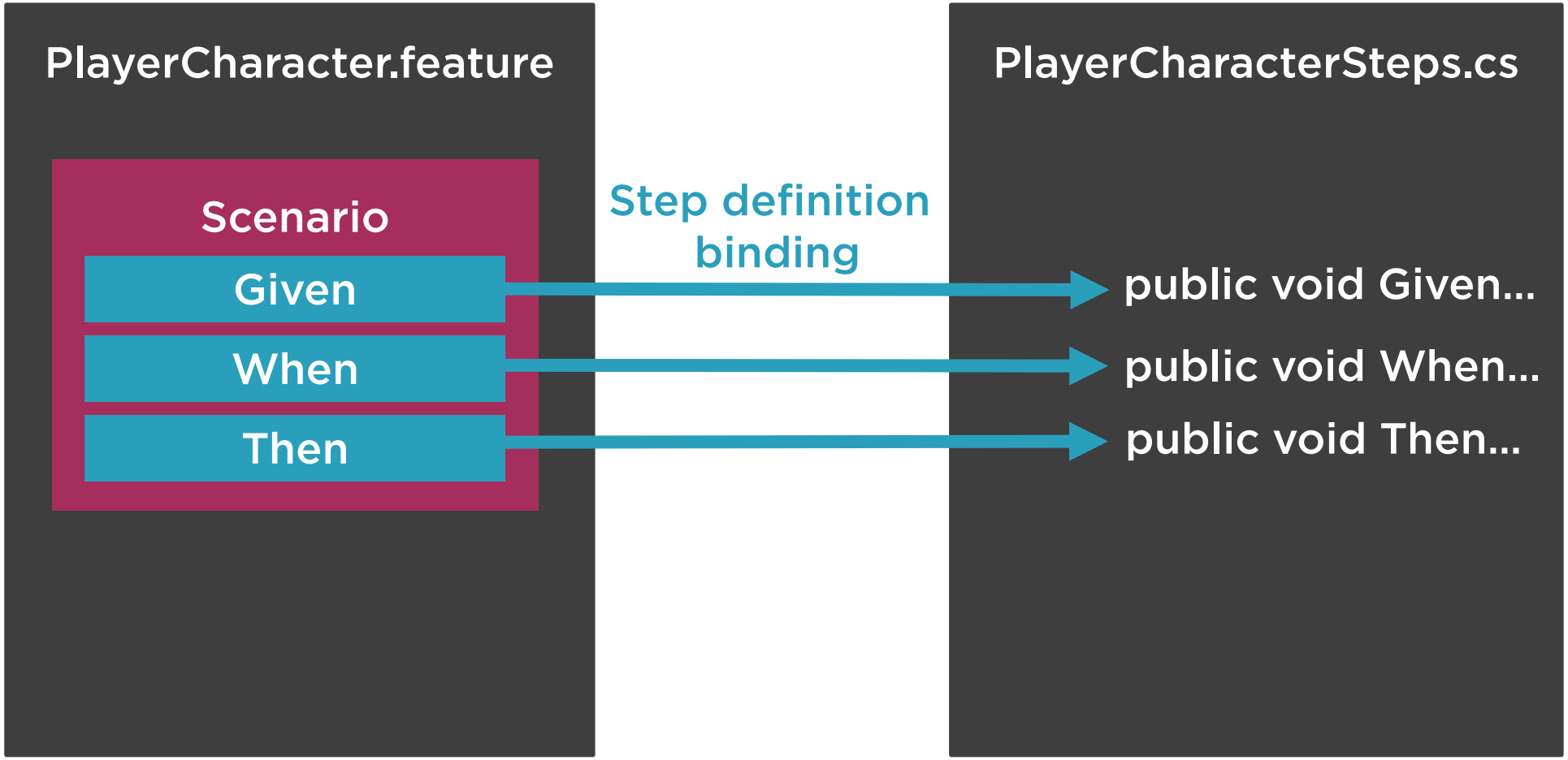
```
@players
Feature: PlayerCharacter
    In order to play the game
    As a human player
    I want my character attributes to be correctly represented
```

Feature Level Tags

Apply at feature level

Inherited by all scenarios in file

Step Definition Code



Step Definition Binding Styles

Regular Expression	Method Naming Convention
Use attribute on method to bind	**Method name matches convention**
Level of indirection	**Underscore or Pascal case**
Method names can be anything	**Method names match step text**
Refactoring of method names	**Harder refactoring**
Extra attribute code	**Less attribute code**

Underscore Step Binding Style

```
[Given]
public void Given_I_m_a_new_player()
{
    _player = new PlayerCharacter();
}
```

Pascal Case Step Binding Style

```
[Given]
public void GivenIMANewPlayer()
{
    _player = new PlayerCharacter();
}
```

Regular Expression Step Binding Style

```
[Given(@"I'm a new player")]
public void GivenIMANewPlayer()
{
    _player = new PlayerCharacter();
}
```

Regular Expression Step Binding Style

```
[Given(@"I'm a new player")]
public void Xyz()
{
    _player = new PlayerCharacter();
}
```

Installation Overview

SpecFlow
Visual Studio
Extension

IDE Integration

Generate step definitions

Editor autocomplete

SpecFlow
NuGet
Packages

SpecFlow package

SpecFlow testing framework integration

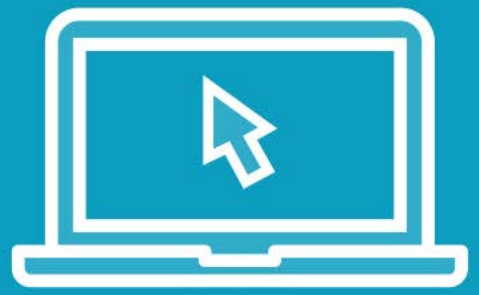
Testing
Framework
NuGet
Packages

Separate testing framework

Used for writing test code

E.g. Assert.Equal

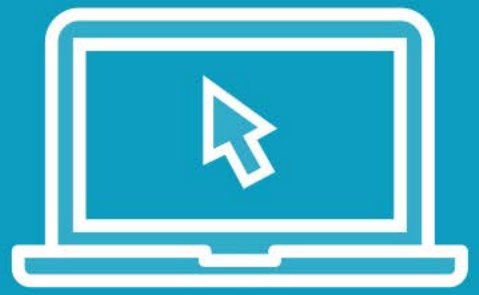
Demo



Getting Started in Visual Studio

- Create GameCore.Specs test project
- Install SpecFlow Visual Studio extension
- Install SpecFlow.xUnit NuGet package
- Install xunit.runner.visualstudio NuGet package
- Add SpecFlow feature file

Demo



Step Definition Binding Styles

- Using SpecFlow tool options
- Generate step definitions
- Regular expression style
- Underscore style
- Pascal case style
- Previewing step definitions
- Copying step code to clipboard

Summary

- Feature headers (“who”, “what”, “why”)
- Scenario steps (“given”, “when”, “then”)
- #Comments and @tags in feature files
- Step definition code maps to scenario steps
- Installation overview and NuGet packages
- Getting started in Visual Studio
- Generate steps definition class
- Copy and preview step definition code
- Regular expression, underscore, and Pascal case styles

Next:

Writing Basic SpecFlow Tests