Add Attributes to Unit Tests

In my previous blog posts, I introduced you to creating unit tests with Visual Studio. The following is the list of blog posts published thus far.

- Introduction to Unit Testing in Visual Studio
- Avoid Hard-Coding in Unit Tests
- Unit Test Initialization and Cleanup

You have seen a few different attributes such as [TestClass], [TestMethod], and [TestInitialize] used to decorate classes and methods. There are several more attributes that you should be aware of. You may or may not use all the attributes presented in this blog post, but you may have a need for them at some time or another.

DataSource Attribute

The unit test framework in Visual Studio can data-drive your unit tests. This means you can read data from a data store and execute a single test repeatedly using the data read in. A later blog post will discuss how to use data-driven tests, so this attribute is not covered here.

Description Attribute

Add a Description attribute to any unit test method to describe why you wrote a particular unit test. It is recommended that you use good, long, description name for the unit test method name as well as the Description attribute. This attribute is not used by the unit test framework, nor does it show up anywhere within the Test Explorer window.

```
[Description("Check to see if a file exists.")]
public void FileNameDoesExist() {
}

[Description("Check to see if file does not exist.")]
public void FileNameDoesNotExist() {
}

[Description("Check for a thrown ArgumentNullException.")]
public void
  FileNameNullOrEmpty_ThrowsArgumentNullException () {
}

[Description("Check for a thrown ArgumentNullException
              using ExpectedException.")]
public void FileNameNullOrEmpty_
   ThrowsArgumentNullException_UsingAttribute () {
}
```

DeploymentItem Attribute

You may add as many DeploymentItem attributes as you need to specify files and folders to copy to the directory where the unit test runs. The DeploymentItem attribute accepts one or two parameters. The first parameter is a folder or a file name. The path is always relative the build output folder. The second parameter, if passed, is to a new path to copy the data in the first parameter. This second path can be relative to the output folder, or can be an absolute path. It is highly recommended you use a relative folder to allow tests to run seamlessly on different machines.

For our simple example, you do not need to use any DeploymentItem attributes, but below is a sample that would copy two files named DeploymentFile1.txt and DeploymentFile2.txt to the build output folder. It is assumed these two files already exist. If the folder or file does not exist, no error is thrown when running in Visual Studio, the test simply continues. However, if you use the command line utility, a warning is generated.

```
[DeploymentItem("DeploymentFile1.txt")]
[DeploymentItem("DeploymentFile2.txt")]
public void FileNameDoesExist() {

}
```

Ignore Attribute

The Ignore attribute is intended to be a temporary attribute you add to skip one or more unit test methods.

```
[Ignore]
public void FileNameDoesExist() {

}
```

Owner Attribute

The Owner attribute allows you to specify the name of the developer responsible for the unit test. This helps with the assignment of work items to the developer of a unit test that breaks.

```
[Owner("PaulS")]
public void FileNameDoesExist() {
}

[Owner("PaulS")]
public void FileNameDoesNotExist() {
}

[Owner("JohnK")]
public void
  FileNameNullOrEmpty_ThrowsArgumentNullException () {
}

[Owner("JohnK")]
public void FileNameNullOrEmpty_
   ThrowsArgumentNullException_UsingAttribute () {
}
```

After the test runs, right mouse click on the test results in the Text Explorer window and select the Group By menu (Figure 1).

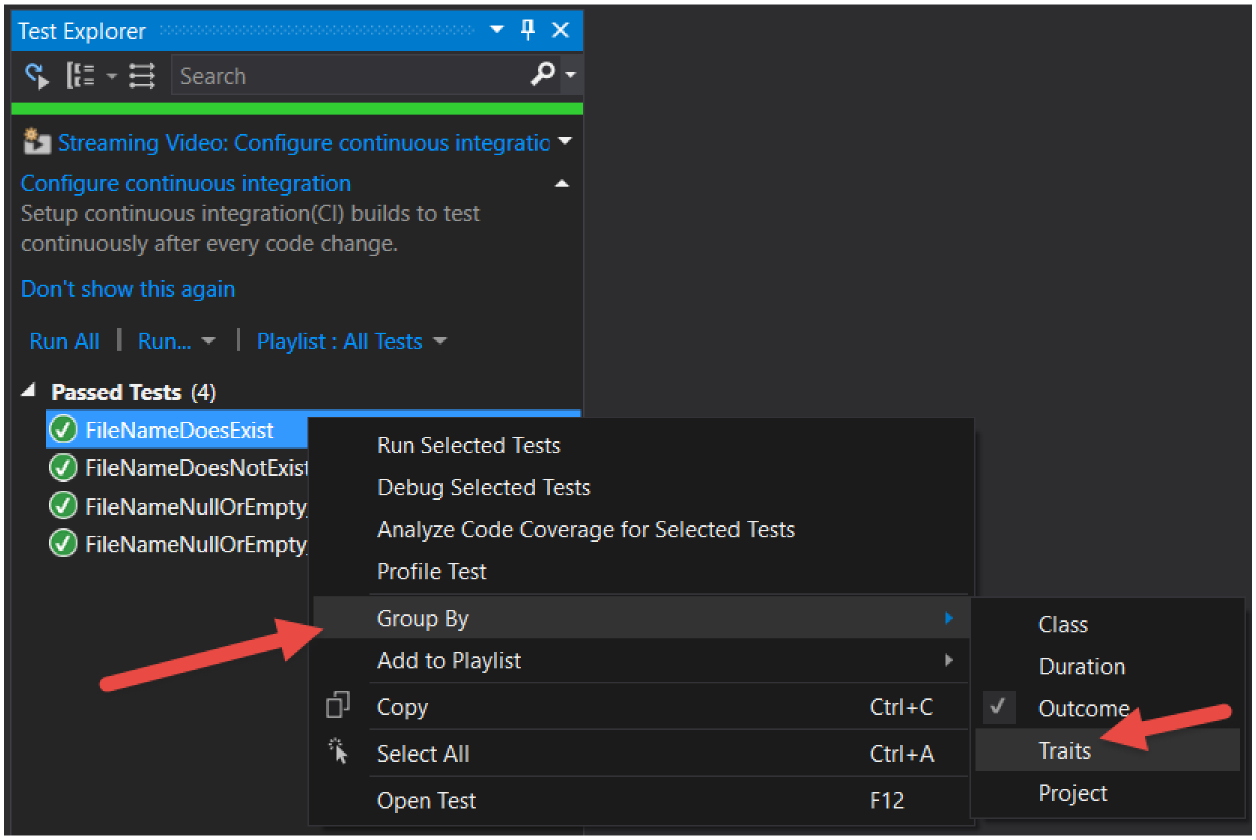
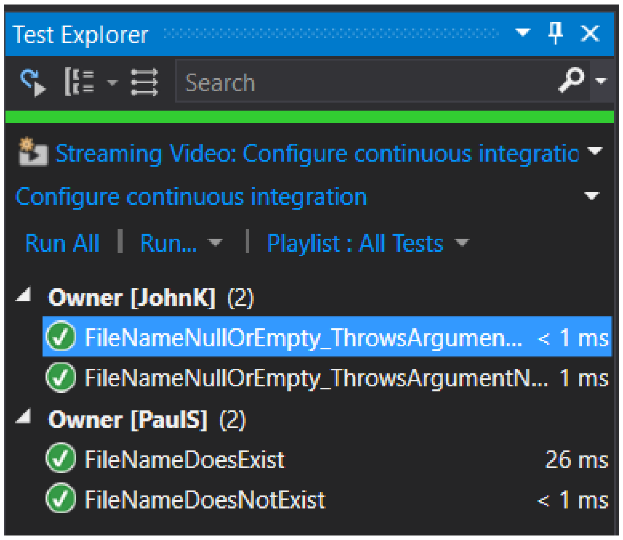


Figure 1: Right mouse click to group by traits

Next select the Traits menu to sort by any attributes you have added (Figure 2).


Test Explorer

Streaming Video: Configure continuous integratio

Configure continuous integration

Run All | Run... | Playlist : All Tests

Owner [JohnK] (2)
FileNameNullOrEmpty_ThrowsArgumen... < 1 ms
FileNameNullOrEmpty_ThrowsArgumentN... 1 ms
Owner [PaulS] (2)
FileNameDoesExist 26 ms
FileNameDoesNotExist < 1 ms

Figure 2: The Test Explorer window can group results by different attributes you add.

Priority Attribute

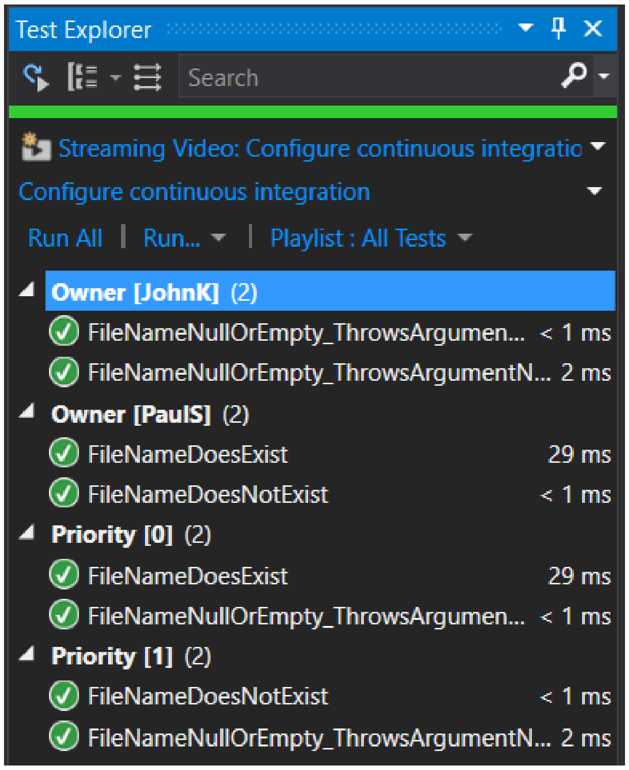
The Priority attribute, like the Owner attribute, is considered a “Trait” by the Test Explorer window. It is not used by the unit test framework itself, but can be grouped in the Test Explorer window (Figure 3). When using the VSTest.Console.exe command-line utility, you may filter the tests to run by the Priority attribute.

```
[Priority(0)]
public void FileNameDoesExist() {
}

[Priority(1)]
public void FileNameDoesNotExist() {
}

[Priority(1)]
public void
  FileNameNullOrEmpty_ThrowsArgumentNullException () {
}

[Priority(0)]
public void FileNameNullOrEmpty_
   ThrowsArgumentNullException_UsingAttribute () {
}
```


Test Explorer
Search
Streaming Video: Configure continuous integratio
Configure continuous integration
Run All
Run...
Playlist : All Tests

```
Owner [JohnK] (2)
  FileNameNullOrEmpty_ThrowsArgumen...  < 1 ms
  FileNameNullOrEmpty_ThrowsArgumentN...  2 ms
Owner [PaulS] (2)
  FileNameDoesExist                      29 ms
  FileNameDoesNotExist                  < 1 ms
Priority [0] (2)
  FileNameDoesExist                      29 ms
  FileNameNullOrEmpty_ThrowsArgumen...  < 1 ms
Priority [1] (2)
  FileNameDoesNotExist                  < 1 ms
  FileNameNullOrEmpty_ThrowsArgumentN...  2 ms
```

Figure 3: Tests can display under multiple traits.

TestCategory Attribute

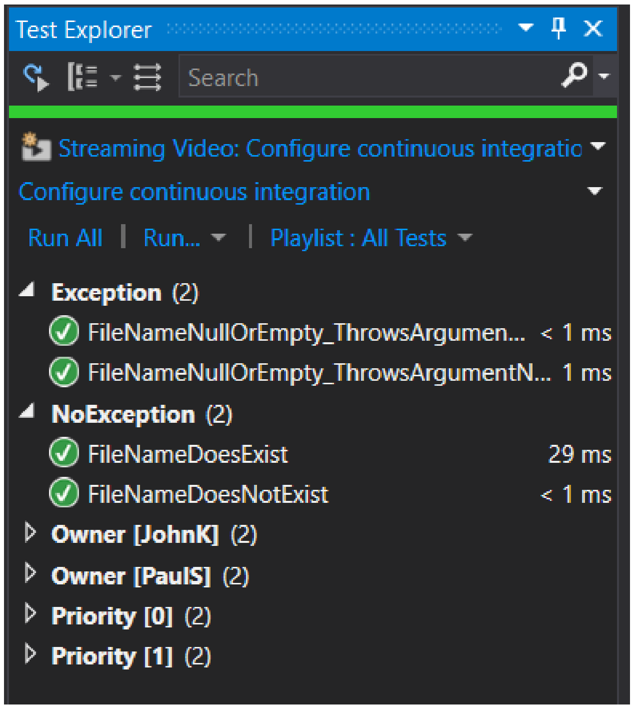
The TestCategory attribute, like Owner and Priority, is a “Trait” you may group upon in the Test Explorer window (Figure 4). You define any category name you wish and assign that name to one or many tests. After all tests have run, you may filter and sort within the Test Explorer window on the category names.

```
[TestCategory("NoException")]
public void FileNameDoesExist() {
}

[TestCategory("NoException")]
public void FileNameDoesNotExist() {
}

[TestCategory("Exception")]
public void
  FileNameNullOrEmpty_ThrowsArgumentNullException () {
}

[TestCategory("Exception")]
public void FileNameNullOrEmpty_
   ThrowsArgumentNullException_UsingAttribute () {
}
```


Test Explorer
Search
Streaming Video: Configure continuous integratic
Configure continuous integration
Run All | Run... | Playlist : All Tests

```
Exception (2)
    FileNameNullOrEmpty_ThrowsArgumen...  < 1 ms
    FileNameNullOrEmpty_ThrowsArgumentN...  1 ms
NoException (2)
    FileNameDoesExist                      29 ms
    FileNameDoesNotExist                  < 1 ms
Owner [JohnK] (2)
Owner [PaulS] (2)
Priority [0] (2)
Priority [1] (2)
```

Figure 4: Categories are common traits to use besides Owner.

Timeout Attribute

Use the Timeout attribute to specify how long a specific method is allowed to run before the unit test framework kills the test and marks it as a failure. The value you specify is expressed in milliseconds. In the following example, the unit test framework will allow the method to execute for only 5 seconds before it will stop the execution and throw an AssertFailedException.

```
[Timout(5000)]
public void FileNameNullOrEmpty_
   ThrowsArgumentNullException_UsingAttribute () {
}
```

Summary

In this blog post you learned about many of the attributes you may apply to unit test classes and methods. The most common attributes you should add are Description, Owner and TestCategory. Description is optional, and should be used in combination with a descriptive unit test method name. The Owner, Priority and TestCategory attributes may be grouped within the Test Explorer window.

Sample Code

You can download the code for this sample at www.pdsa.com/downloads. Choose the category “PDSA Blogs”, then locate the sample **Add Attributes to Unit Tests**.