# Unit Testing Using the Command Line

This is another in my series of blog posts on unit testing. If you are not familiar with unit testing, go back and read these posts.

- Introduction to Unit Testing in Visual Studio
- Avoid Hard-Coding in Unit Tests
- Unit Test Initialization and Cleanup
- Add Attributes to Unit Tests
- Using Assert Classes and Methods in Unit Tests

In this post, you are going to learn to run unit tests from the command line. This allows you to schedule tests using task manager or any other automated scheduling tool. The VSTest.Console.exe is the tool you use as a .NET developer to run your unit tests in your .NET test dll.

## Simple VSTest.Console Run

The VSTest.Console.exe is typically installed into the following path: C:\Program Files (x86)\Microsoft Visual Studio 14.0\Common7\IDE\CommonExtensions\Microsoft\TestWindow. When you run this tool, you need to prefix this path to the .exe, or add this path to the PATH environment variable. If you open a **Developer Command Prompt for VS2015** (Figure 1) this path will already be included.

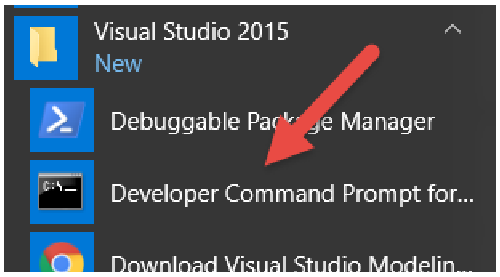
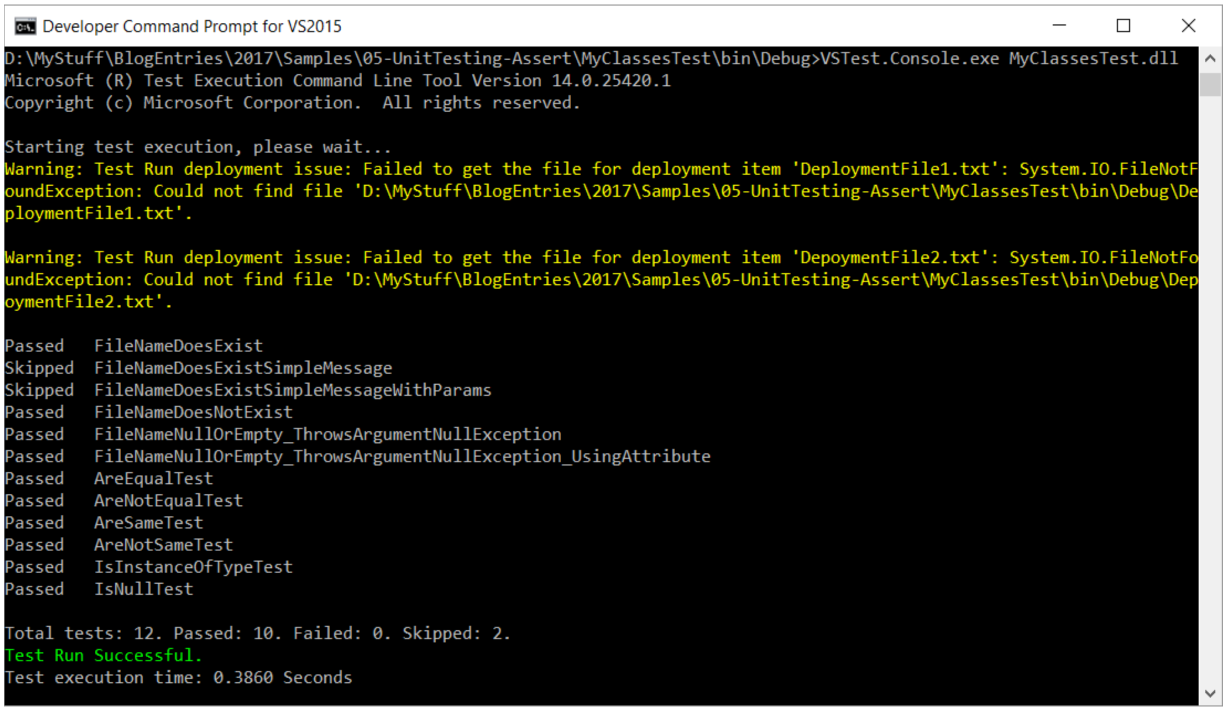

Visual Studio 2015
New
Debuggable Package Manager
Developer Command Prompt for...
Download Visual Studio Modelin...

Figure 1: Use the Developer Command Prompt

If you have been creating the project as you have been following along with this series of blog posts, you have a DLL named MyClassesTest.dll located in the \bin\Debug folder of where your project is located on your disk. Open a Developer Command Prompt and navigate to that folder. Type in the following in your command window.

```
VSTest.Console.exe MyClassesTest.dll
```

Press the Enter key and you should see something that looks like Figure 2. You may or may not have the warnings depending on if you added the DeploymentItem attributes.



```
D:\MyStuff\BlogEntries\2017\Samples\05-UnitTesting-Assert\MyClassesTest\bin\Debug>VSTest.Console.exe MyClassesTest.dll
Microsoft (R) Test Execution Command Line Tool Version 14.0.25420.1
Copyright (c) Microsoft Corporation.  All rights reserved.

Starting test execution, please wait...
Warning: Test Run deployment issue: Failed to get the file for deployment item 'DeploymentFile1.txt': System.IO.FileNotF
oundException: Could not find file 'D:\MyStuff\BlogEntries\2017\Samples\05-UnitTesting-Assert\MyClassesTest\bin\Debug\De
ploymentFile1.txt'.

Warning: Test Run deployment issue: Failed to get the file for deployment item 'DepoymentFile2.txt': System.IO.FileNotFo
undException: Could not find file 'D:\MyStuff\BlogEntries\2017\Samples\05-UnitTesting-Assert\MyClassesTest\bin\Debug\Dep
oymentFile2.txt'.

Passed   FileNameDoesExist
Skipped  FileNameDoesExistSimpleMessage
Skipped  FileNameDoesExistSimpleMessageWithParams
Passed   FileNameDoesNotExist
Passed   FileNameNullOrEmpty_ThrowsArgumentNullException
Passed   FileNameNullOrEmpty_ThrowsArgumentNullException_UsingAttribute
Passed   AreEqualTest
Passed   AreNotEqualTest
Passed   AreSameTest
Passed   AreNotSameTest
Passed   IsInstanceOfTypeTest
Passed   IsNullTest

Total tests: 12. Passed: 10. Failed: 0. Skipped: 2.
Test Run Successful.
Test execution time: 0.3860 Seconds
```

Figure 2: A simple run of the VSTest.Console

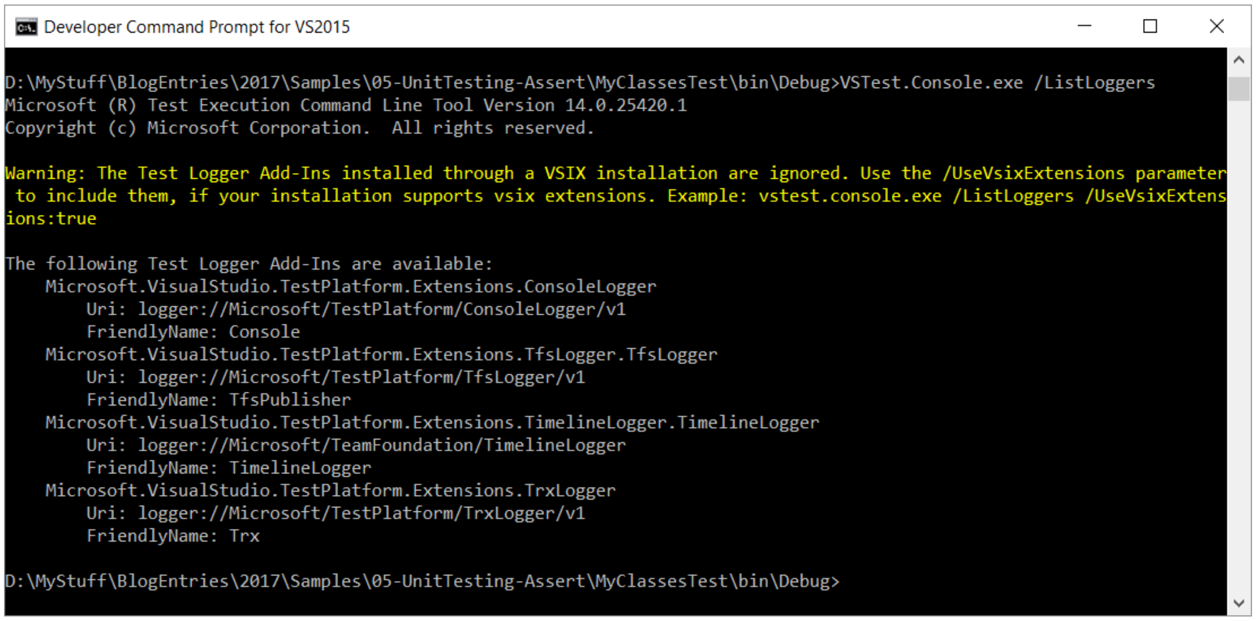
## View Installed Loggers

Now that you know how to run your unit tests from the command line, you now need to learn to log the results to a file that you can look at later. If you are going to be running your unit tests overnight, you want to come back in the morning to see the results. If they are just sitting in a command window, you could accidentally close that window and you would lose the results.

Instead, the VSTest.Console utility has a set of loggers that you can use. Type in the following in the command window.

```
VSTest.Console.exe /ListLoggers
```

Press the Enter key to see a screen that looks like Figure 3.



Developer Command Prompt for VS2015

```
D:\MyStuff\BlogEntries\2017\Samples\05-UnitTesting-Assert\MyClassesTest\bin\Debug>VSTest.Console.exe /ListLoggers
Microsoft (R) Test Execution Command Line Tool Version 14.0.25420.1
Copyright (c) Microsoft Corporation.  All rights reserved.

Warning: The Test Logger Add-Ins installed through a VSIX installation are ignored. Use the /UseVsixExtensions parameter
 to include them, if your installation supports vsix extensions. Example: vstest.console.exe /ListLoggers /UseVsixExtens
ions:true

The following Test Logger Add-Ins are available:
    Microsoft.VisualStudio.TestPlatform.Extensions.ConsoleLogger
        Uri: logger://Microsoft/TestPlatform/ConsoleLogger/v1
        FriendlyName: Console
    Microsoft.VisualStudio.TestPlatform.Extensions.TfsLogger.TfsLogger
        Uri: logger://Microsoft/TestPlatform/TfsLogger/v1
        FriendlyName: TfsPublisher
    Microsoft.VisualStudio.TestPlatform.Extensions.TimelineLogger.TimelineLogger
        Uri: logger://Microsoft/TeamFoundation/TimelineLogger
        FriendlyName: TimelineLogger
    Microsoft.VisualStudio.TestPlatform.Extensions.TrxLogger
        Uri: logger://Microsoft/TestPlatform/TrxLogger/v1
        FriendlyName: Trx

D:\MyStuff\BlogEntries\2017\Samples\05-UnitTesting-Assert\MyClassesTest\bin\Debug>
```

Figure 3: View the log options you can use with VSTest.Console

The ConsoleLogger is the one you are currently looking at in your command window. The TfsLogger is useful if you are using Team Foundation Server as it allows you to send the results to TFS so you can assign work items based on any failed unit tests. The last one is the one that will be useful if you do not have TFS. The TrxLogger creates a .trx file which you can load into Visual Studio and see a list of all of your unit tests. You can then click on each test and see the results, and the outputs for that test.

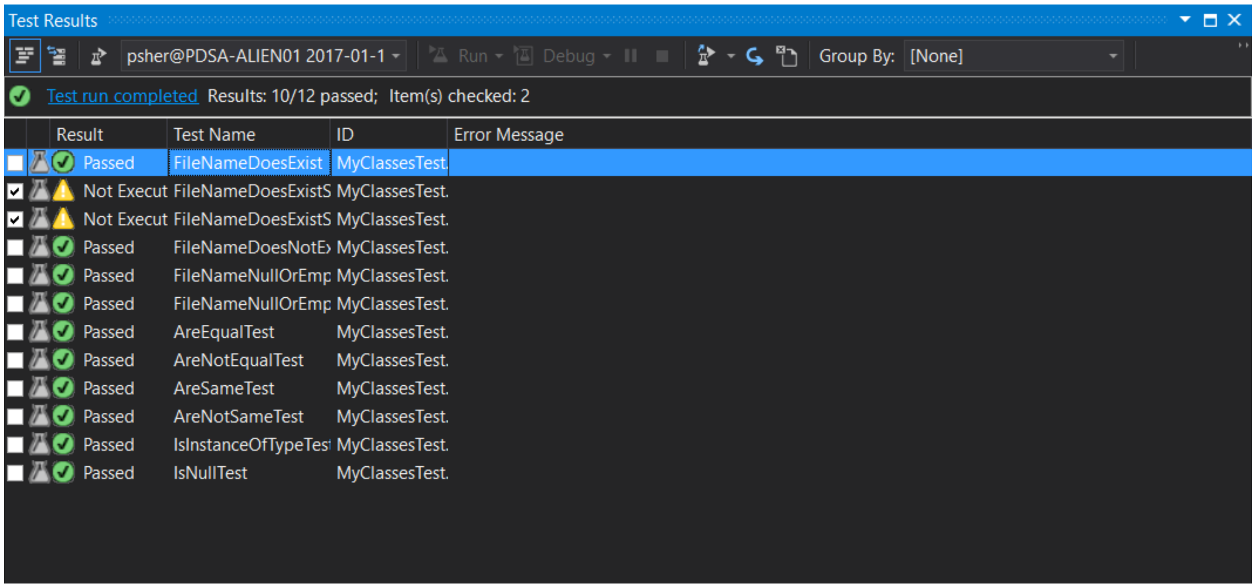
## Using Logger

Let's take a look at using the TrxLogger option when using the VSTestConsole.exe utility. Type in the following into the command window.

```
VSTest.Console.exe MyClassesTest.dll /Logger:trx
```

Press the Enter key and the unit testing will run. The results are now stored in a .trx file located in the TestResults folder under your \bin\Debug folder. Each time you run a new .trx file is added with a later date and time add to the file name.

Double click on any of the *.trx files and it will load the results into a Test Results window in Visual Studio as shown in Figure 4. You can double-click on any of the tests to see the output from that test.



Test run completed Results: 10/12 passed; Item(s) checked: 2

| | Result | Test Name | ID | Error Message |
|---|---|---|---|---|
| | Passed | FileNameDoesExist | MyClassesTest. | |
| ✓ | Not Execut | FileNameDoesExistS | MyClassesTest. | |
| ✓ | Not Execut | FileNameDoesExistS | MyClassesTest. | |
| | Passed | FileNameDoesNotE› | MyClassesTest. | |
| | Passed | FileNameNullOrEmp | MyClassesTest. | |
| | Passed | FileNameNullOrEmp | MyClassesTest. | |
| | Passed | AreEqualTest | MyClassesTest. | |
| | Passed | AreNotEqualTest | MyClassesTest. | |
| | Passed | AreSameTest | MyClassesTest. | |
| | Passed | AreNotSameTest | MyClassesTest. | |
| | Passed | IsInstanceOfTypeTest | MyClassesTest. | |
| | Passed | IsNullTest | MyClassesTest. | |

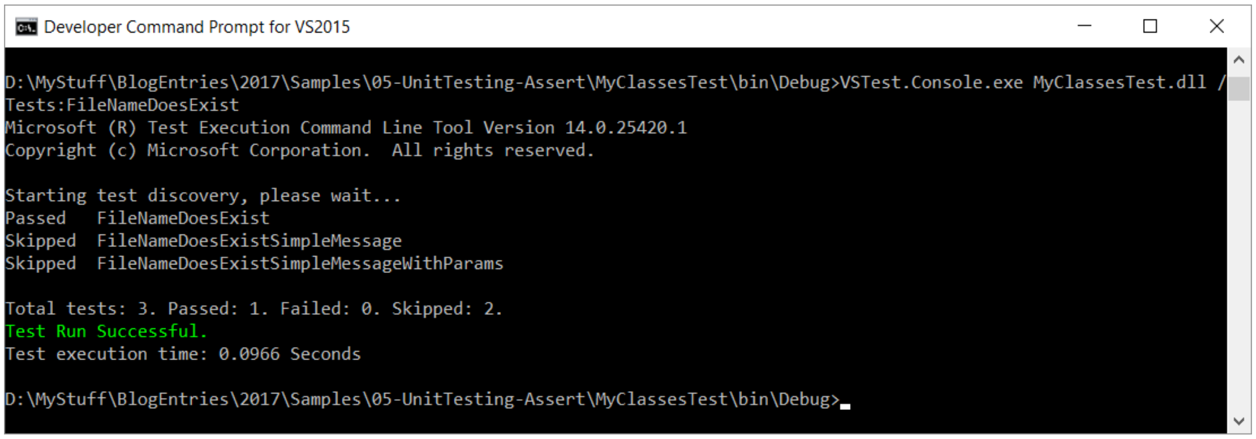
Figure 4: Visual Studio can display all results from a .trx file

# Run Specific Test(s)

The VSTest.Console utility allows you to specify single or multiple test method names to run. If you are just testing one or two items, there is no reason to run all the tests in your DLL. Add the /Tests parameter on the command line followed by a comma-delimited list of method names you wish to execute. Type the following into your command window.

```
VSTest.Console.exe MyClassesTest.dll /Tests:FileNameDoesExist
```

Press the Enter key and you should see a result that looks like Figure 5.



```
D:\MyStuff\BlogEntries\2017\Samples\05-UnitTesting-Assert\MyClassesTest\bin\Debug>VSTest.Console.exe MyClassesTest.dll /
Tests:FileNameDoesExist
Microsoft (R) Test Execution Command Line Tool Version 14.0.25420.1
Copyright (c) Microsoft Corporation.  All rights reserved.

Starting test discovery, please wait...
Passed   FileNameDoesExist
Skipped  FileNameDoesExistSimpleMessage
Skipped  FileNameDoesExistSimpleMessageWithParams

Total tests: 3. Passed: 1. Failed: 0. Skipped: 2.
Test Run Successful.
Test execution time: 0.0966 Seconds

D:\MyStuff\BlogEntries\2017\Samples\05-UnitTesting-Assert\MyClassesTest\bin\Debug>
```

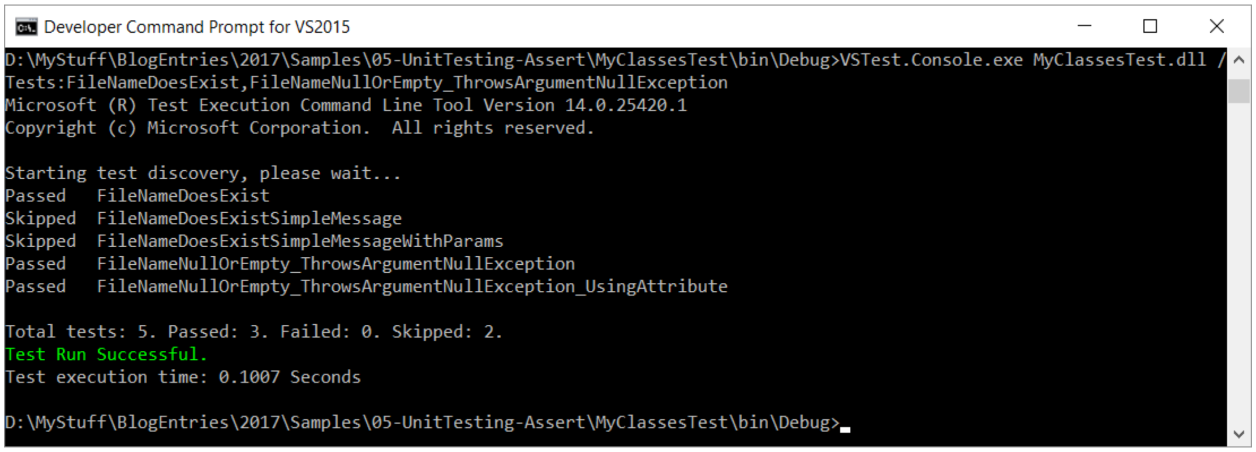
Figure 5: The /Tests parameter does pattern matching on your method names

Notice that multiple tests where run even though you only specified a single name. This is because the /Tests parameter uses pattern matching on your method names. It will find any method that starts with the name you pass in and run those methods.

You can use a comma-delimited list to specify different sets of methods to run. Type the following into your command window.

```
VSTest.Console.exe MyClassesTest.dll
   /Tests:FileNameDoesExist,
    FileNameNullOrEmpty_ThrowsArgumentNullException
```

Press the Enter key and you should see results that look similar to Figure 6.



```
D:\MyStuff\BlogEntries\2017\Samples\05-UnitTesting-Assert\MyClassesTest\bin\Debug>VSTest.Console.exe MyClassesTest.dll /
Tests:FileNameDoesExist,FileNameNullOrEmpty_ThrowsArgumentNullException
Microsoft (R) Test Execution Command Line Tool Version 14.0.25420.1
Copyright (c) Microsoft Corporation.  All rights reserved.

Starting test discovery, please wait...
Passed   FileNameDoesExist
Skipped  FileNameDoesExistSimpleMessage
Skipped  FileNameDoesExistSimpleMessageWithParams
Passed   FileNameNullOrEmpty_ThrowsArgumentNullException
Passed   FileNameNullOrEmpty_ThrowsArgumentNullException_UsingAttribute

Total tests: 5. Passed: 3. Failed: 0. Skipped: 2.
Test Run Successful.
Test execution time: 0.1007 Seconds

D:\MyStuff\BlogEntries\2017\Samples\05-UnitTesting-Assert\MyClassesTest\bin\Debug>
```

Figure 6: You may use a comma-delimited list after the /Tests parameter

## Filter Tests to Run based on Attributes

As mentioned in the blog post on attributes, the Priority attribute is not used by the unit test framework. However, when you use the VSTest.Console utility, you are allowed to filter based on the Priority attribute. Type in the following to the command window and you can run just those methods that have the Priority(1) attribute.

```
VSTest.Console.exe MyClassesTest.dll
   /TestCaseFilter:"Priority=1"
```

The /TestCaseFilter lets you specify attributes and specific names of methods to run. For example, if you want to just run one test with the name of FileNameDoesExist, you type in the following into the command window to run that one test.

```
VSTest.Console.exe MyClassesTest.dll
   /Name:"FileNameDoesExist"
```

Another attribute you can specify to run is TestCategory. Run the following in the command window to just run those tests marked with [TestCategory("NoException")].

```
VSTest.Console.exe MyClassesTest.dll
   /TestCaseFilter:"TestCategory=NoException"
```

You are not allowed to use both the /TestCaseFilter parameter and the /Tests parameter together. You must just run one or the other.

## Summary

Running unit tests in a batch is made easy with the VSTest.Console.exe utility. This utility allows you to log the output to a file for later review. You can also log to TFS to help you assign work items to unit tests that fail. The VSTest.Console utility also lets you filter the tests to run using either a /Tests parameter, or a /TestCaseFilter parameter.

# Sample Code

You can download the code for this sample at www.pdsa.com/downloads. Choose the category “PDSA Blogs”, then locate the sample **Unit Testing Using the Command Line**.