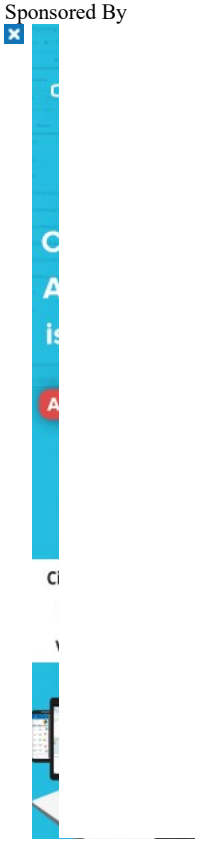
Command Line: Using dotnet watch test for continuous testing with .NET Core 1.0 and XUnit.net

March 27, '17 Posted in DotNetCore | Open Source



I've installed .NET Core 1.0 on my machine. Let's see if I can get a class library and tests running and compiling automatically using only the command line. (Yes, some of you are freaked out by my (and other folks') appreciation of a nice, terse command line. Don't worry. You can do all this with a mouse if you want. I'm just enjoying the CLI.

> NOTE: This is considerably updated from the project.json version in 2016.

First, I installed from http://dot.net/core. This should all work on Windows, Mac, or Linux.

```
C:\> md testexample & cd testexample

C:\testexample> dotnet new sln
Content generation time: 33.0582 ms
The template "Solution File" created successfully.

C:\testexample> dotnet new classlib -n mylibrary -o mylibrary
Content generation time: 40.5442 ms
The template "Class library" created successfully.

C:\testexample> dotnet new xunit -n mytests -o mytests
Content generation time: 87.5115 ms
The template "xUnit Test Project" created successfully.

C:\testexample> dotnet sln add mylibrary\mylibrary.csproj
Project `mylibrary\mylibrary.csproj` added to the solution.

C:\testexample> dotnet sln add mytests\mytests.csproj
Project `mytests\mytests.csproj` added to the solution.

C:\testexample> cd mytests

C:\testexample\mytests> dotnet add reference ..\mylibrary\mylibrary.csproj
Reference `..\mylibrary\mylibrary.csproj` added to the project.

C:\testexample\mytests> cd ..

C:\testexample> dotnet restore
  Restoring packages for C:\Users\scott\Desktop\testexample\mytests\mytests.csproj...
  Restoring packages for C:\Users\scott\Desktop\testexample\mylibrary\mylibrary.csproj...
  Restore completed in 586.73 ms for C:\Users\scott\Desktop\testexample\mylibrary\mylibrary.csproj.
  Installing System.Diagnostics.TextWriterTraceListener 4.0.0.
...SNIP...
  Installing Microsoft.NET.Test.Sdk 15.0.0.
  Installing xunit.runner.visualstudio 2.2.0.
  Installing xunit 2.2.0.
  Generating MSBuild file C:\Users\scott\Desktop\testexample\mytests\obj\mytests.csproj.nuget.g.props.
```

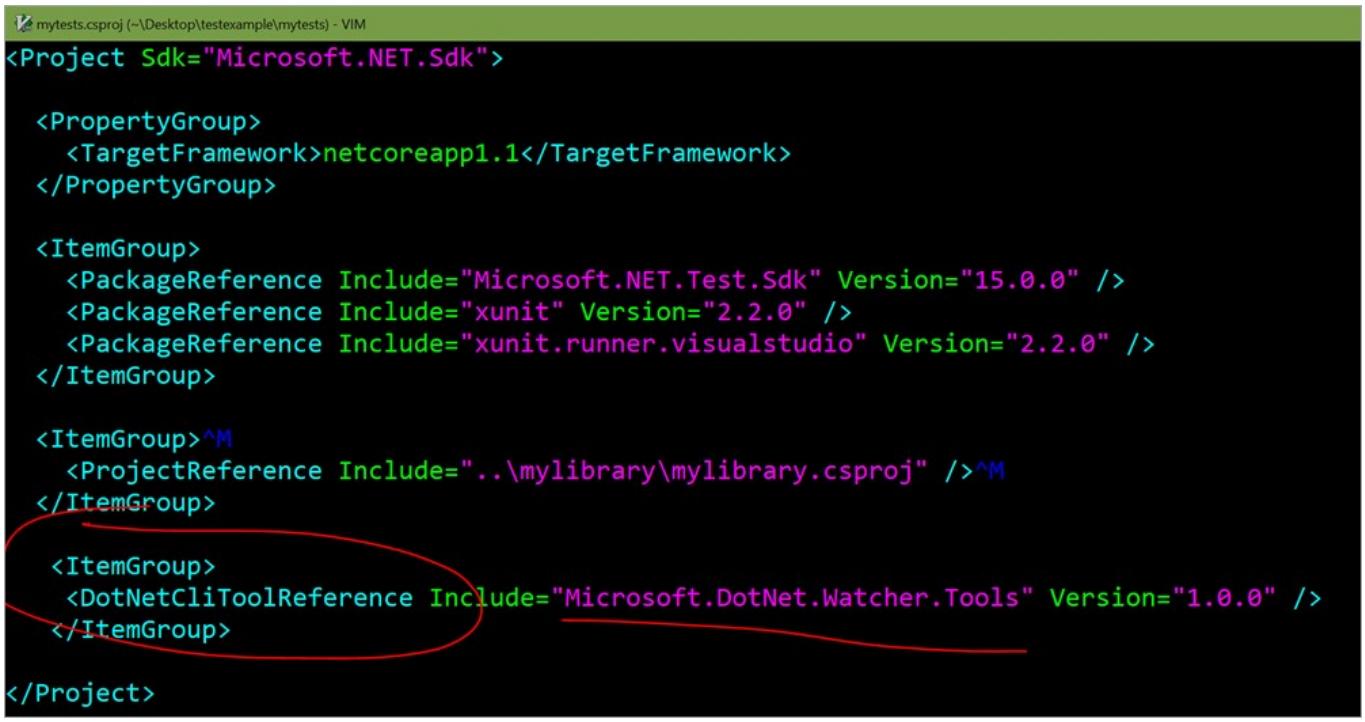
```
  Generating MSBuild file C:\Users\scott\Desktop\testexample\mytests\obj\mytests.csproj.nuget.g.targets.
  Writing lock file to disk. Path: C:\Users\scott\Desktop\testexample\mytests\obj\project.assets.json
  Installed:
      16 package(s) to C:\Users\scott\Desktop\testexample\mytests\mytests.csproj

C:\testexample> cd mytests & dotnet test

Build started, please wait...
Build completed.

Test run for C:\testexample\mytests\bin\Debug\netcoreapp1.1\mytests.dll(.NETCoreApp,Version=v1.1)
Microsoft (R) Test Execution Command Line Tool Version 15.0.0.0
Copyright (c) Microsoft Corporation.  All rights reserved.

Starting test execution, please wait...
[xUnit.net 00:00:00.5539676]   Discovering: mytests
[xUnit.net 00:00:00.6867799]   Discovered:  mytests
[xUnit.net 00:00:00.7341661]   Starting:    mytests
[xUnit.net 00:00:00.8691063]   Finished:    mytests

Total tests: 1. Passed: 1. Failed: 0. Skipped: 0.
Test Run Successful.
Test execution time: 1.8329 Seconds
```

Of course, I'm testing nothing yet but pretend there's a test in the tests.cs and something it's testing (that's why I added a reference) in the library.cs, OK?

Now I want to have my project build and tests run automatically as I make changes to the code. I can't "dotnet add tool" yet so I'll add this line to my test's project file:

```
<ItemGroup>
 <DotNetCliToolReference Include="Microsoft.DotNet.Watcher.Tools" Version="1.0.0" />
</ItemGroup>
```

Like this:



Then I just dotnet restore to bring in the tool.

In order to start the tests, I don't write **dotnet test**, I run "**dotnet watch test**." The main command is watch, and then WATCH calls TEST. You can also dotnet watch run, etc.

> NOTE: There's a color bug using only cmd.exe so on "DOS" you'll see some ANSI chars. That should be fixed in a minor release soon - the PR is in and waiting. On bash or PowerShell things look fin.

In this screenshot, you can see as I make changes to my test and hit save, the DotNetWatcher Tool sees the change and restarts my app, recompiles, and re-runs the tests.

> Test Run Successful

All this was done from the command line. I made a solution file, made a library project and a test project, made the test project reference the library, then built and ran the tests. If I could add the tool from the command line I wouldn't have had to manually touch the project file at all.

Again, to be sure, all this is stuff you can (and do) do in Visual Studio manually all the time. But I'll race you anytime. ;)

About Scott

Scott Hanselman is a former professor, former Chief Architect in finance, now speaker, consultant, father, diabetic, and Microsoft employee. He is a failed stand-up comic, a cornrower, and a book author.

Disclaimer: The opinions expressed herein are my own personal opinions and do not represent my employer's view in any way.