Visual Studio Code for .Net Framework

Ask Question

I am having difficulty figuring out if and how I can use Visual Studio **Code** to develop and debug command-line/console/libraries of C#.Net programs which **can not run on .Net Core**, i.e. they **require .Net Framework**. I need to access Oracle which do not have a .Net Core provider but it does have a Managed .Net Framework provider. I use VS 2015/2017 for this task but would like to switch to VS Code if I could code, build and debug .Net Framework target C# programs. I have tried Google search and could not find anything.

.net visual-studio-code

asked Dec 8 '17 at 3:05
Acid Rider
83 • 10

Check out this question posted here stackoverflow.com/questions/29953743/… – Matthew Pigram Dec 8 '17 at 3:10

I saw this, it does not answer my question. If you think it does, please cut and paste the relevant paragraph here. Thank you. – Acid Rider Dec 8 '17 at 3:29

have you tried using the oracle provider with .net core? – Neville Nazerane Dec 8 '17 at 3:57

i tried using .net provider with core, got lots of exceptions with various dll dependencies, plus its not supported so i gave up..... the system.data.odbc works in Core but very slow. – Acid Rider Dec 8 '17 at 19:33

3 Answers

Ok, although I am late, I am posting this answer, as it might help other people facing this problem, which was quite exhausting for me.

First thing, the more recent updates for Visual Studio Code do support building and debugging projects for .NET Framework, but it is very limited:

> The C# extension supports limited full .NET framework debugging. It can only debug 64-bit applications with portable PDBs. https://github.com/OmniSharp/omnisharp-vscode/wiki/Desktop-.NET-Framework

As it says in OmniSharp (responsible for C# extension) github, the project must be a 64-bit application with portable PDBs.

But, even after reading many issues and discussions about this topic, it remained a little bit unclear for me what were the necessary steps. So I will expose here a little guide with the steps that I have followed and that worked for me, and hopefully, will also work for you.

1) The necessary files/folders are:

a) .vscode with launch.json and tasks.json

b) bin\Debug folder for your exe application and the assemblies you might want to create a reference to

d) the .csproj and Program.cs files

e) optionally a batch file, whose purpose I will describe later

2) Install the MSBuild 15

3) In the .csproj file:

- change the Project Sdk="Microsoft.NET.Sdk" to Project ToolsVersion="15.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003"
- in the first Property Group we must set the OutputType to Exe (the default may be dll), remove the TargetFramework property, replacing with TargetFrameworkVersion with value v4.6.1 (example for .NET Framwork 4.6.1, it may be 4.7 for instance), and finally put the runtimes win-x64 and win7-x64 (and any other that the compiler may complain). This first Group should look like this:

```
<PropertyGroup>
   <OutputType>Exe</OutputType>
   <TargetFrameworkVersion>v4.6.1</TargetFrameworkVersion>
   <RuntimeIdentifiers>win-x64;win7-x64</RuntimeIdentifiers>
</PropertyGroup>
```

Run code snippet Expand snippet

- set another property group with the following items:

```
  <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Debug|AnyCPU' ">
    <PlatformTarget>x64</PlatformTarget>
    <DebugSymbols>true</DebugSymbols>
    <DebugType>portable</DebugType>
    <Optimize>false</Optimize>
    <OutputPath>bin\Debug\</OutputPath>
    <DefineConstants>DEBUG;TRACE</DefineConstants>
    <ErrorReport>prompt</ErrorReport>
    <WarningLevel>4</WarningLevel>
  </PropertyGroup>
```

Run code snippet Expand snippet

Some comments: the condition used signals that these properties only apply when the configuration passed to the compiler is Debug and the Platform is "AnyCPU", you might want to insert other conditions with different values, or even dont use a condition at all; the most import values here are: The **PlatformTarget** property **must** be **x64** and the **DebugType** must be **portable**; the output path is set to bin\Debug.

- As we are not using the Microsoft Sdk we must include the Program.cs, so that the compiler can find it:

```
  <ItemGroup>
    <Compile Include="Program.cs" />
  </ItemGroup>
```

Run code snippet Expand snippet

- create the necessary references to your project; example:

```
<ItemGroup>
    <Reference Include="mscorlib" />
    <Reference Include="System.Core" />
    <Reference Include="System.Windows" />
    <Reference Include="System.ServiceModel" />
    <Reference Include="System.Net" />
    <Reference Include="System.Xml" />
    <Reference Include="System" />
    <Reference Include="System.Xml.Linq" />
    <Reference Include="System.Data.DataSetExtensions" />
    <Reference Include="Microsoft.CSharp" />
    <Reference Include="System.Data" />
    <Reference Include="System.Net.Http" />
</ItemGroup>
```

Run code snippet Expand snippet

- finally import the following tools (make sure you follow the order exposed here, placing this in the beginning for instance you generate an error)

```
  <Import Project="$(MSBuildToolsPath)\Microsoft.CSharp.targets" />
```

Run code snippet Expand snippet

The whole thing should look like this:

```
<Project ToolsVersion="15.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">

  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFrameworkVersion>v4.6.1</TargetFrameworkVersion>
    <RuntimeIdentifiers>win-x64;win7-x64</RuntimeIdentifiers>
  </PropertyGroup>

  <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Debug|AnyCPU' ">
    <PlatformTarget>x64</PlatformTarget>
    <DebugSymbols>true</DebugSymbols>
    <DebugType>portable</DebugType>
    <Optimize>false</Optimize>
    <OutputPath>bin\Debug\</OutputPath>
    <DefineConstants>DEBUG;TRACE</DefineConstants>
    <ErrorReport>prompt</ErrorReport>
    <WarningLevel>4</WarningLevel>
  </PropertyGroup>

  <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Release|AnyCPU' ">
    <PlatformTarget>x64</PlatformTarget>
    <DebugType>portable</DebugType>
    <Optimize>true</Optimize>
    <OutputPath>bin\Release\</OutputPath>
    <DefineConstants>TRACE</DefineConstants>
    <ErrorReport>prompt</ErrorReport>
    <WarningLevel>4</WarningLevel>
  </PropertyGroup>

  <ItemGroup>
    <Compile Include="Program.cs" />
  </ItemGroup>

  <ItemGroup>
    <Reference Include="mscorlib" />
    <Reference Include="System.Core" />
    <Reference Include="System.Windows" />
```

```
    <Reference Include="System.ServiceModel" />
    <Reference Include="System.Net" />
    <Reference Include="System.Xml" />
    <Reference Include="System" />
    <Reference Include="System.Xml.Linq" />
    <Reference Include="System.Data.DataSetExtensions" />
    <Reference Include="Microsoft.CSharp" />
    <Reference Include="System.Data" />
    <Reference Include="System.Net.Http" />
  </ItemGroup>

  <Import Project="$(MSBuildToolsPath)\Microsoft.CSharp.targets" />

</Project>
```

Run code snippet Expand snippet

4) in the launch.json:

- Create a new configuration(eg MyLauncher), whose **type must be clr**, and that points to your program; the preLaunchTask will be set to a manually configured one -"mybuild" for instance - that will be specified in the tasks.json; an example of configuration is:

```
   "version": "0.2.0",
   "configurations": [

       {
           "name": "MyLauncher",
           "type":"clr",
           "request": "launch",
           "preLaunchTask": "mybuild",
           "program": "${workspaceFolder}/bin/Debug/<project>.exe",
           "args":[],
           "console": "internalConsole",
           "stopAtEntry": false,
           "internalConsoleOptions": "openOnSessionStart"
       },

       { other configurations...
       }

    ,]
}
```

Run code snippet Expand snippet

5) in the tasks.json:

- Create a task "mybuild" with the commands to build your project
- We will use the MSBuild 15 here (dont use the dotnet build - at least it has not worked for me)
- You can directly point to the (path)\MSBuild.exe (or msbuild.exe, if it is in the %PATH%) file with the arguments to build the project. One example is shown below, note that Ive set the Configuration to Debug and the platform to AnyCPU, matching the condition Ive set in the .csproj file, also note that the backslashes in \"AnyCPU\" are because of the use of the quotation marks.

```
{
    "version": "2.0.0",
    "tasks": [

        {
            "label": "mybuild",
            "command":"<path to msbuild>\MSBuild.exe",
            "type":"shell",
            "args":[
                "<project>.csproj",
                "/t:Build",
                "/p:Configuration=Debug",
                "/p:Platform=\"AnyCPU\""
            ]
        }

    ]
}
```

Run code snippet Expand snippet

- but there is another way, using the .bat file; in my case the path to the MSBuild.exe had spaces and that was generating an error when the task run, so that I've put the following code in a .bat file (save notepad as name.bat):

  "(path)\MSBuild.exe" (project).csproj /t:Build /p:Configuration=Debug /p:Platform="AnyCPU"

And then set the "mybuild" task to:

```
        {
            "label": "mybuild",
            "command":"build.bat",
            "type":"shell",
            "args":[]
        }
```

Run code snippet Expand snippet

Where build.bat is the batch file I have created with the previous code

6) After this, you might have to save, close and reopen the files (this many times fixes problems for me)

7) Set your configuration in the debugger to MyLauncher:

print

8) Run your code with the green play button; it will call the MyLaunch, that first will build your project with MSBuild 15 and then run the exe file

So that was it.

Hope it helps you.

It would be great if you could give any feedback.

Regards

```
I dont know how to use this site very well, so Ive let some references here, because it
wasnt letting me:

https://github.com/OmniSharp/omnisharp-vscode/issues/813
https://github.com/OmniSharp/omnisharp-vscode/issues/1365
https://github.com/OmniSharp/omnisharp-vscode/wiki/Portable-PDBs
```

Run code snippet Expand snippet

edited Apr 13 at 21:00

answered Apr 13 at 20:46

Mario Figueiredo

21 ● 3

Unfortunately, it doesn't feature intellisense for C/C++, only syntax highlighting: code.visualstudio.com/docs/languages EDIT: no debugger integration for C/C++ either. The git integration is really nice though! Seems more designed for web applications, the debugger works for node.js

Whilst this does not specify C#, it stands to reason that the same standards apply (which is that there is no debugger and no compile functionality).

Quote taken from comment on first answer of What exactly is Visual Studio Code?

answered Dec 8 '17 at 3:39

Matthew Pigram

703 ● 1 ● 15 ● 45

according to YouTube videos on VS Code, C/C++ is supported and has debugger support. In any case, I have the answer, .Net Framework C# and VS Code do not mix well as of Dec 2017. Pity. We move on. – Acid Rider Dec 9 '17 at 8:47

https://code.visualstudio.com/docs/languages/csharp

Quote:

Note: VS Code does not support **debugging** applications running on the Desktop .NET **Framework**.

It looks like Visual Studio 'full-fat' IDE remains a requirement for .Net Framework. Major pity.

answered Dec 8 '17 at 3:49

Acid Rider

83 ● 10

There is limited support for 64 bit executables with portable PDB files. There is more info at the link: github.com/Microsoft/vscode-docs/issues/1339 – Cameron Jan 26 at 0:04