How to perform automatic obfuscation using Visual Studio 2013/15?

NOVEMBER 7, 2015 / ANSHUL SOJATIA / 5 COMMENTS / 5 MIN. READ

Obfuscation is one of the most essential aspect that any experienced programmer would consider before releasing the executable files to the client. After all you wouldn't want any of your hard work (read written code) to be stolen or modified without your permission, would you? With obfuscation, the distributable file is modified in such a way that it's becomes harder (if not impossible) for a cracker to reverse engineering your code. You can read more about obfuscation on Wikipedia.

Microsoft Visual Studio too comes with its native basic obfuscator called DotFuscator which is good for beginners. But when you are going commercial, you might want to look for some other available options. Some are free or open source while other ones may reduce some weight of your wallet 😉

No matter which ones you use, you have to manually pass the dll or exe file to the obfuscator executable along with all the dependencies. Though this is manageable for small projects or solutions with few exe's or dll's, it starts becoming a laborious experience as the size of your project grows. More executable files are created and you start manual obfuscation for each of those individual files. And then at some point, you would think if this process can be automated.

Forget Manual, Go for Automation

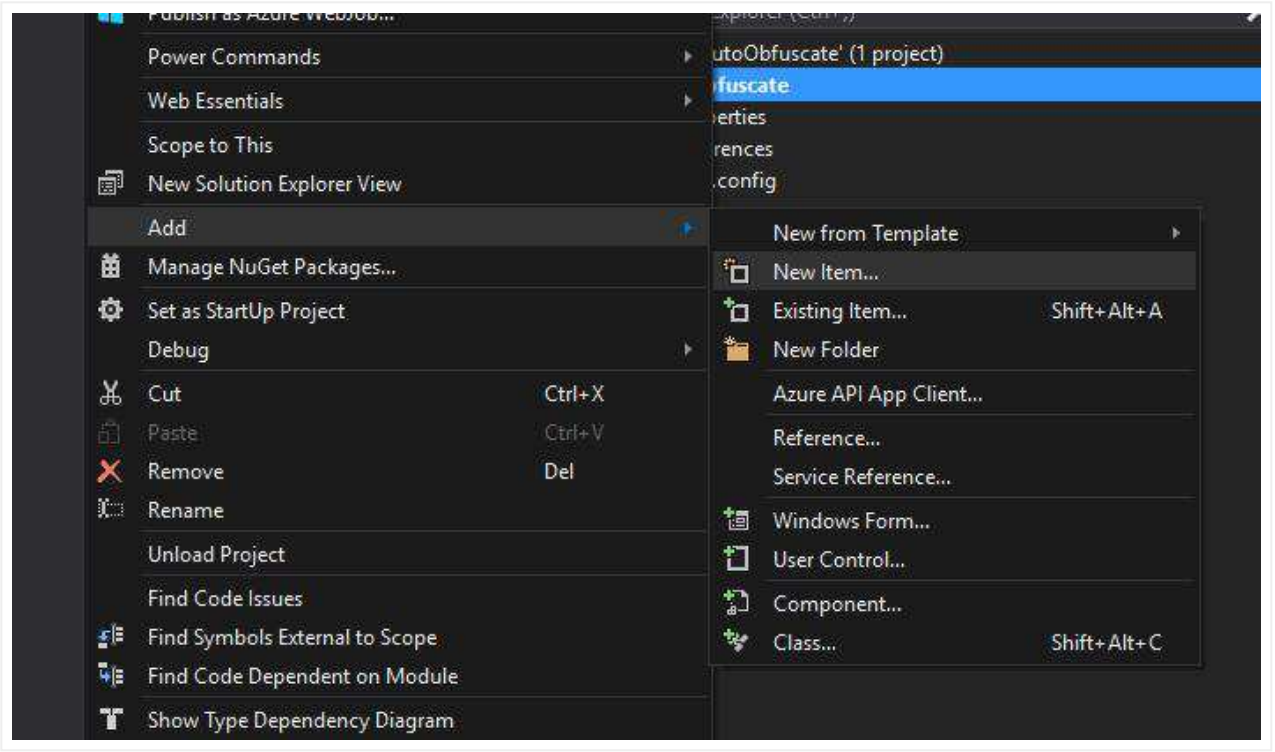
In this tutorial, I'll show you how you can automate the process of obfuscation. The tutorial uses ConfuserEx obfuscator which is pretty good for commercial projects. And the best part is it's free to use.

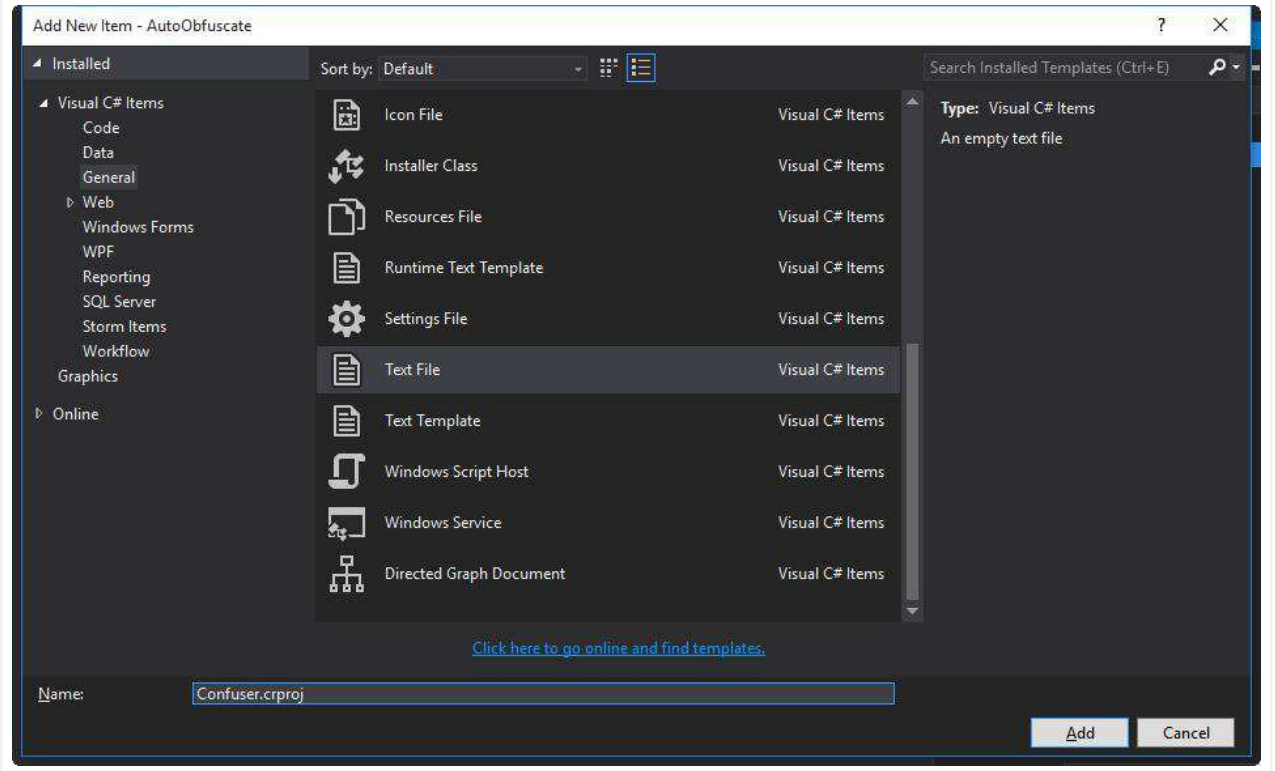
The following is a list of tools which we'll be using.

- Visual Studio 2013 Community (The process should be same for Visual Studio 2015 as well)
- ConfuserEx (You can download it from GitHub). We'll use the command line version for our purpose.
- Windows PowerShell

To automate the process of obfuscation, Visual Studio provides us with awesome Pre-Build and Post-Build events that we'll make use of. So here we go.

Step 1: Create the confuser project file





In your Visual Studio solution, right click on the project and add a new text file from the 'General' tree view node. Name the file as confuser.crproj. This will be an xml file that we'll pass to the confuser.

Add the following code to this file and save it.

```
<project outputDir="{OutputDir}" baseDir="{BaseDir}" xmlns="http://confuser.codeplex.cor
```

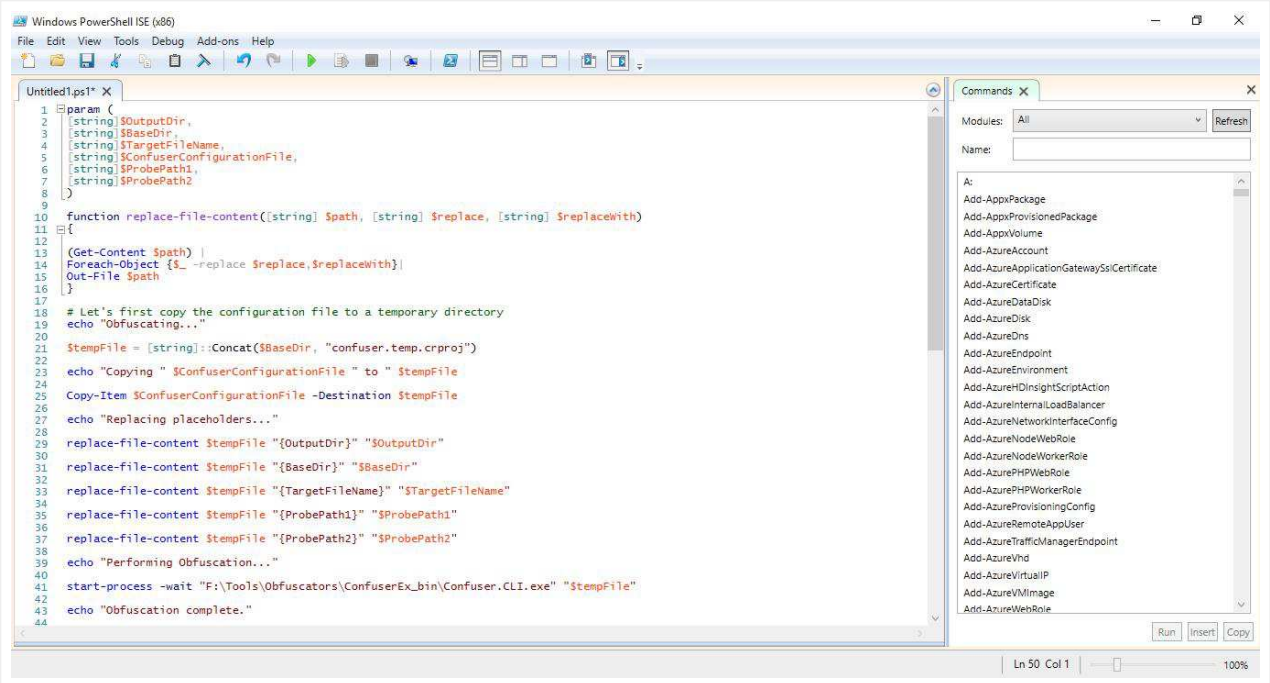
```
  <rule pattern="true" preset="aggressive" />
  <module path="{TargetFileName}">
    <rule pattern="true" preset="aggressive" />
  </module>
  <probePath>{ProbePath1}</probePath>
  <probePath>{ProbePath2}</probePath>
</project>
```

This file contains all the information that's required by confuser to perform it's operation. Notice the placeholder values that we have put in curly braces {}. This will be explained in further steps. You can easily create this file by opening Confuser.

Depending on your requirements and level of obfuscation you want, you can set the value of preset from aggressive to some other value. Refer to ConfuserEx documentation for details.

Note: The probePath elements in above file specify the directory where dependencies or references of your projects are present. If you don't have any external dependency in your project, you can safely remove these elements. Conversely, if you have more than two directories where dependencies are stored, you can add more elements. Just remember to change their placeholders accordingly.

Step 2: Create PowerShell Script



To execute any external application from Visual Studio, windows powershell can be used. You can read more about powershell in this great tutorial by David Frette.

To create a powershell script, you can either use notepad or any other editor. But the real power comes with Windows Powershell Integrated Scripting Environment (ISE). So we'll use ISE for our powershell script.

Open PowerShell ISE and add the following code to this file.

```
param (
[string]$OutputDir,
[string]$BaseDir,
[string]$TargetFileName,
[string]$ConfuserConfigurationFile,
[string]$ProbePath1,
[string]$ProbePath2
)

function replace-file-content([string] $path, [string] $replace, [string] $replaceWith)
{
(Get-Content $path) |
Foreach-Object {$_ -replace $replace,$replaceWith}|
Out-File $path
}

# Let's first copy the configuration file to a temporary directory
echo "Obfuscating..."
$tempFile = [string]::Concat($BaseDir, "confuser.temp.crproj")
echo "Copying " $ConfuserConfigurationFile " to " $tempFile
Copy-Item $ConfuserConfigurationFile -Destination $tempFile
echo "Replacing placeholders..."
replace-file-content $tempFile "{OutputDir}" "$OutputDir"
replace-file-content $tempFile "{BaseDir}" "$BaseDir"
replace-file-content $tempFile "{TargetFileName}" "$TargetFileName"
replace-file-content $tempFile "{ProbePath1}" "$ProbePath1"
replace-file-content $tempFile "{ProbePath2}" "$ProbePath2"
echo "Performing Obfuscation..."
start-process -wait "F:ToolsObfuscatorsConfuserEx_binConfuser.CLI.exe" "$tempFile"
echo "Obfuscation complete."
echo "Removing " $tempFile
Remove-Item $tempFile
echo "Done!!!"
```

Save this file on your harddisk at an easy accessible path. I had mine saved at **D:confuserps.ps1**. We'll require this file in next step.

What does this code do?

This script accepts a few parameters as explained:

$OutputDir – The directory where obfuscated file will be saved

$BaseDir – The directory where unobfuscated executable is stored

$TargetFileName – The file name of target obfuscated file

$ConfuserConfigurationFile – The configuration file for confuser

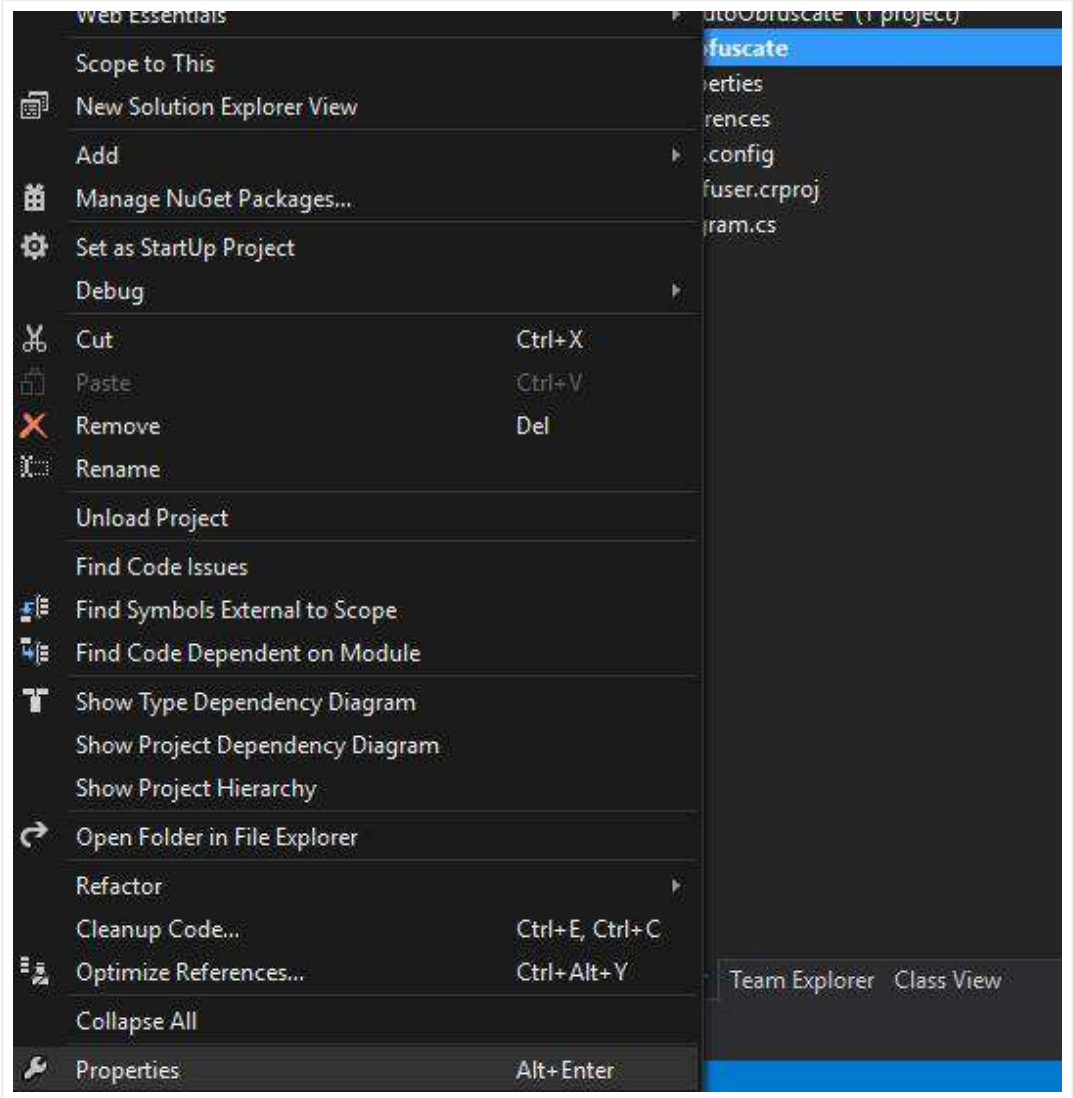
$ProbePath1, $ProbePath2… – The paths where confuser should look for dependencies of executable

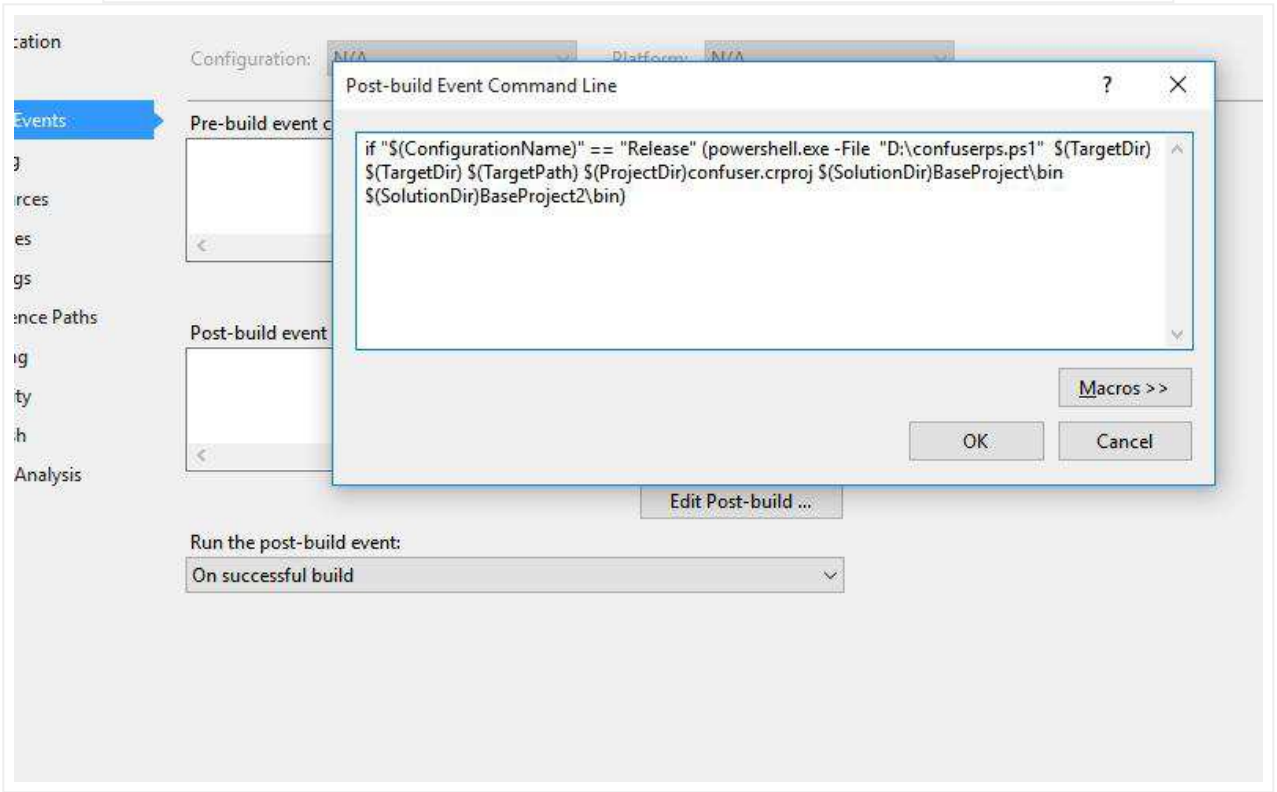
Next it creates a function to replace the contents of text file. The confuser project file is then copied from the Visual Studio Project Directory to the Output Directory. Subsequent lines perform actual replacement of placeholders in this temporarily copied file.

This file is then passed to the command line version of ConfuserEx.

Note: The probePath parameter in above file specify the directory where dependencies or references of your projects are present. If you don't have any external dependency in your project, you can safely remove these parameters. Conversely, if you have more than two directories where dependencies are stored, you can add more parameters. Just remember to change their replacements accordingly.

Step 3: Setup Visual Studio project to use these files


Web Essentials
Scope to This
New Solution Explorer View
Add
Manage NuGet Packages...
Set as StartUp Project
Debug
Cut Ctrl+X
Paste Ctrl+V
Remove Del
Rename
Unload Project
Find Code Issues
Find Symbols External to Scope
Find Code Dependent on Module
Show Type Dependency Diagram
Show Project Dependency Diagram
Show Project Hierarchy
Open Folder in File Explorer
Refactor
Cleanup Code... Ctrl+E, Ctrl+C
Optimize References... Ctrl+Alt+Y
Collapse All
Properties Alt+Enter
fuscate
erties
rences
.config
fuser.crproj
ram.cs
Team Explorer
Class View


Configuration:
Post-build Event Command Line
if "$(ConfigurationName)" == "Release" (powershell.exe -File "D:\confuserps.ps1" $(TargetDir) $(TargetDir) $(TargetPath) $(ProjectDir)confuser.crproj $(SolutionDir)BaseProject\bin $(SolutionDir)BaseProject2\bin)
Macros >>
OK
Cancel
Pre-build event
Post-build event
Edit Post-build ...
Run the post-build event:
On successful build

Right click on your Visual Studio project and goto Properties. From the properties page, go to **Build Events** tab.

Click on the **Edit Post-Build...** button. In the post build popup editor, add the following code

```
if "$(ConfigurationName)" == "Release" (powershell.exe -File  "D:confuserps.ps1"  $(Targ
```

What does this code do?

The script first checks if the project is being built in Debug mode or Release mode. Because we don't want to obfuscate the assembly in Debug mode, a check here ensures that the obfuscation is performed only when compilation is done is Release mode.

We then call powershell.exe and pass the PowerShell script file that we created in Step 2. Make sure you replace the file path appropriately as per your path. Following that we pass parameters listed in Step 2 in that order. We use Visual Studio macros to dynamically replace the paths according the project directory. To know more about Visual Studio macros, read this article on MSDN.

Note: The last two parameters specify the probe paths (the paths where confuser will look for dependencies). If your project doesn't have any dependencies, you can safely remove these parameters from Post-Build event as well as from the PowerShell script we created earlier.

Step 4: There is no step 4 🙂

Now change your build mode to Release and build the project. You'll see that after build succeeds, the executable file is automatically obfuscated by Confuser. Your output window will also show the progress of obfuscation as it's performed.

Follow the same procedure for all the projects that you wish to have automatic obfuscation enabled.

One Last Point

If you get a permission error in the output window while the Post-Build event is executed, make sure that you have set the execution policy of PowerShell to Unrestricted.

The process above uses ConfuserEx, however you can use similar process to work with any other obfuscator. Just refer to the documentation of that particular tool.

Related

7 Must Have Tools to Reverse Engineer .NET applications
June 10, 2016
In "Technology"

How to access custom settings in NopCommerce?
June 12, 2015
In "Tutorial"

Move websites from one web host to another without wget
May 6, 2016
In "Tips"

Tutorial

OBFUSCATION SECURITY VISUAL-STUDIO

PREVIOUS POST
How to access custom settings in NopCommerce?

NEXT POST
Automatic entity framework migrations in NopCommerce plugins

Javad Richa • 8 months ago

Thank you for the great article.
What if the project path has many spaces and many folders embedded (bask slash)... this would definitely cause an issue. kindly advise how to escape those characters from VS..
thanks again

▲ | ▼ • Reply • Share ›

ErHardeep Verma • a year ago

Hi Anshul. Very nice & useful info.
1. Will debugging be available after this integration?
2.Will it work on VS 2010 also or it requires only 2013+?

▲ | ▼ • Reply • Share ›

Anshul Sojatia Mod ➚ ErHardeep Verma • a year ago

Thanks a lot Hardeep,
I think the same process should work for VS 2010 as well. However the extension wouldn't probably support VS 2010. **@Avinab Malla** would be able to tell more about that.

▲ | ▼ • Reply • Share ›

Avinab Malla • a year ago

Great guide!
I have implemented the steps outlined here as a Visual Studio extension. This should make life much easier, and it also works with ClickOnce!
You can download it here: https://visualstudiogallery...

▲ | ▼ • Reply • Share ›

Anshul Sojatia Mod ➚ Avinab Malla • a year ago

Hi Avinab,
Thanks a lot for this contribution. This will definitely be a useful extension. Glad that my article was helpful in some way :)

▲ | ▼ • Reply • Share ›

ALSO ON **WITTYLOG**

Case Insensitive Key Dictionary in C#

2 comments • a year ago•

Avatar**Anshul Sojatia** — Hey Cesar, thanks for letting me know. I didn't know about that

Why nopCommerce is a long runner – For Developers

2 comments • a year ago•

Avatar**Anshul Sojatia** — Thanks for your

QUICK LINKS

Home

About

Tutorials

Tips

Off Topic

SOCIAL

TAGS

angularjs asp.net bugs c#.net desktop ecommerce entity-framework fun google inspire love mobile mvc nopcommerce obfuscation productivity security software testing tips url shortener visual-studio web-hosting wordpress