Daily .NET Tips

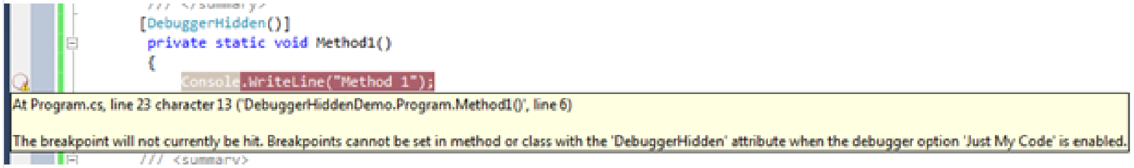
Your Daily Development Productivity Tips & Tricks

Hide Methods from debugger Using DebuggerHidden attribute

By Abhijit Jana | January 31, 2011 3 Comments

DubuggerHidden attribute tells the Visual Studio debugger that the method is hidden from the debugging process and while debugging. This is quite helpful when you don't want to go to stepping inside of a method while debugging.

When you mark a method with DebuggerHidden() attributes, It's explicitly tells the debugger not to step inside of that methods and no break point will be hit over that method. Now I am going to explain the same using a Example with Intellitrace debugging.



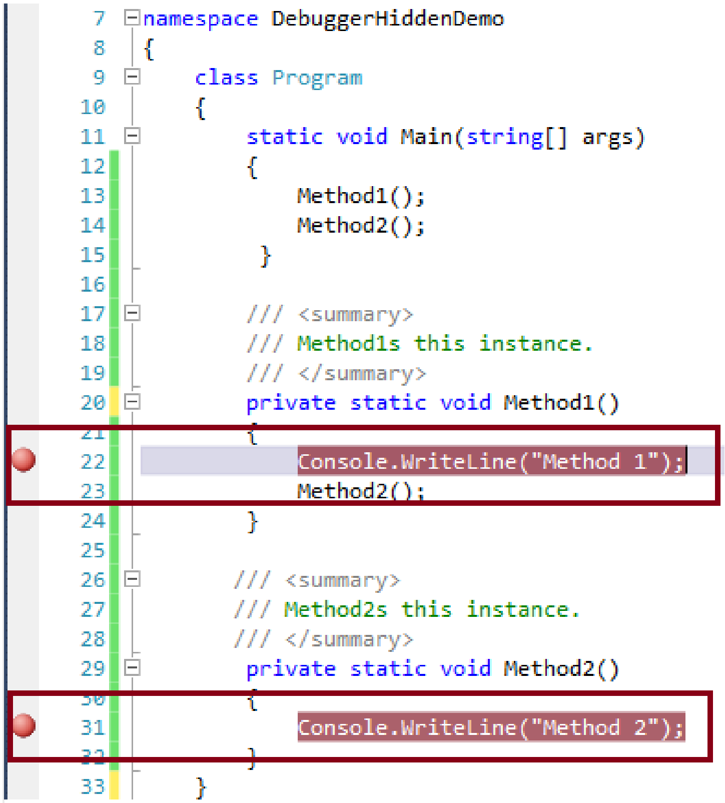
```
        /// </summary>
        [DebuggerHidden()]
         private static void Method1()
         {
             Console.WriteLine("Method 1");
```

At Program.cs, line 23 character 13 ('DebuggerHiddenDemo.Program.Method1()', line 6)

The breakpoint will not currently be hit. Breakpoints cannot be set in method or class with the 'DebuggerHidden' attribute when the debugger option 'Just My Code' is enabled.

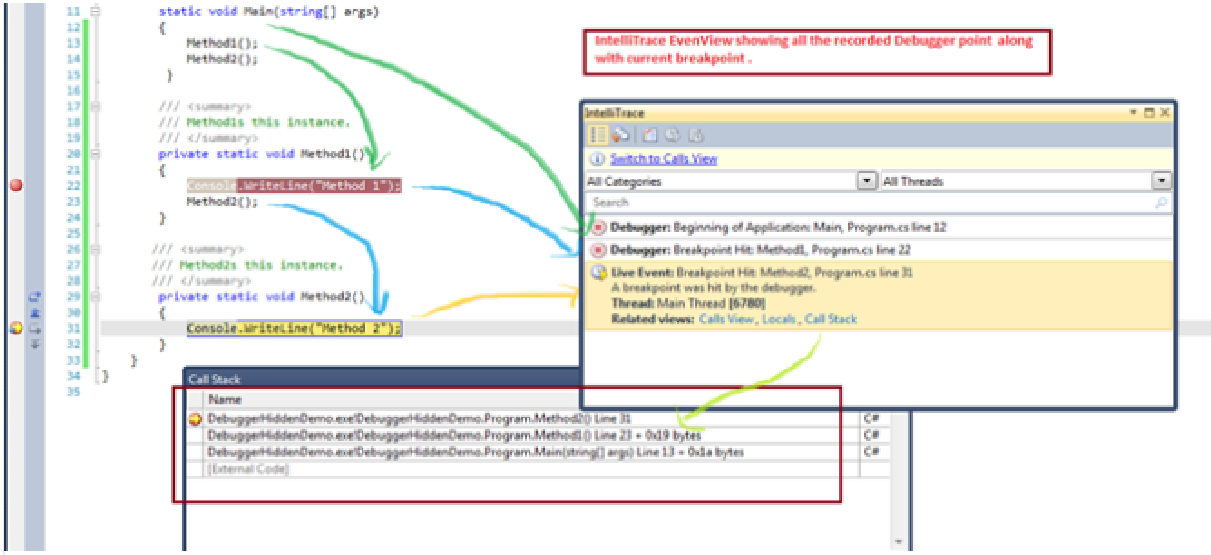
```
        /// <summary>
```

Let's consider you have below code snippet



```
 7  namespace DebuggerHiddenDemo
 8  {
 9      class Program
10      {
11          static void Main(string[] args)
12          {
13              Method1();
14              Method2();
15           }
16
17          /// <summary>
18          /// Method1s this instance.
19          /// </summary>
20          private static void Method1()
21          {
22              Console.WriteLine("Method 1");
23              Method2();
24          }
25
26         /// <summary>
27         /// Method2s this instance.
28         /// </summary>
29          private static void Method2()
30          {
31              Console.WriteLine("Method 2");
32          }
33      }
```

We have set two break points in each of the method (*Line 22*, and *Line 31* as shown) . Now if you run this application and check the output intellitrace debugging window , you will get bellow out put



```
        static void Main(string[] args)
        {
            Method1();
            Method2();
         }

        /// <summary>
        /// Method1s this instance.
        /// </summary>
        private static void Method1()
        {
            Console.WriteLine("Method 1");
            Method2();
        }

       /// <summary>
       /// Method2s this instance.
       /// </summary>
        private static void Method2()
        {
            Console.WriteLine("Method 2");
        }
    }
}
```

IntelliTrace EvenView showing all the recorded Debugger point along with current breakpoint .

IntelliTrace

Switch to Calls View

All Categories | All Threads

Search

Debugger: Beginning of Application: Main, Program.cs line 12

Debugger: Breakpoint Hit: Method1, Program.cs line 22

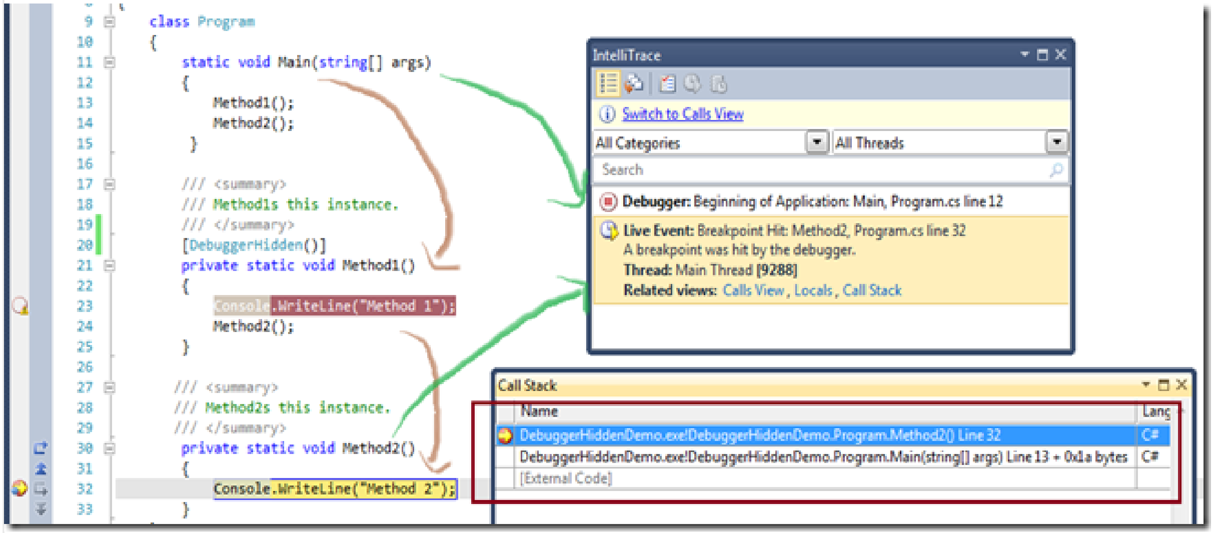
Live Event: Breakpoint Hit: Method2, Program.cs line 31
A breakpoint was hit by the debugger.
Thread: Main Thread [6780]
Related views: Calls View , Locals , Call Stack

Call Stack

Name	
DebuggerHiddenDemo.exe!DebuggerHiddenDemo.Program.Method2() Line 31	C#
DebuggerHiddenDemo.exe!DebuggerHiddenDemo.Program.Method1() Line 23 + 0x19 bytes	C#
DebuggerHiddenDemo.exe!DebuggerHiddenDemo.Program.Main(string[] args) Line 13 + 0x1a bytes	C#
[External Code]	

As per the above image, you can see the flow of the application from Main() > Method1() > Method2() . If you also check the current live event's "Call Stack" , that also represent the call of method2() from method1(). This is as per our expected behavior.

Now, If you don't need to go inside Method1() while debugging and debugger should not stop inside method1() for any of the breakpoint, you have to add "*DebuggerHidden()*" attributes with the *method1().* If you try to debug the application with the same breakpoint, CallStack() and IntelliTrace debugger view will be something different.



```
    class Program
    {
        static void Main(string[] args)
        {
            Method1();
            Method2();
        }

        /// <summary>
        /// Method1s this instance.
        /// </summary>
        [DebuggerHidden()]
        private static void Method1()
        {
            Console.WriteLine("Method 1");
            Method2();
        }

       /// <summary>
       /// Method2s this instance.
       /// </summary>
        private static void Method2()
        {
            Console.WriteLine("Method 2");
        }
```

IntelliTrace

Switch to Calls View

All Categories | All Threads

Search

Debugger: Beginning of Application: Main, Program.cs line 12

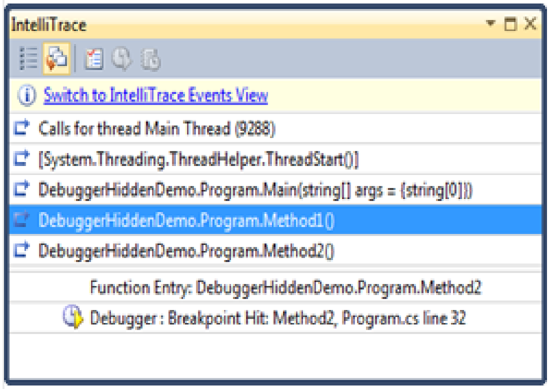
Live Event: Breakpoint Hit: Method2, Program.cs line 32
A breakpoint was hit by the debugger.
Thread: Main Thread [9288]
Related views: Calls View , Locals , Call Stack

Call Stack

Name	Lang
DebuggerHiddenDemo.exe!DebuggerHiddenDemo.Program.Method2() Line 32	C#
DebuggerHiddenDemo.exe!DebuggerHiddenDemo.Program.Main(string[] args) Line 13 + 0x1a bytes	C#
[External Code]	

If you check the above image, debugger reached to Method2() through Method1(), but breakpoint in method1() never hits, because of the “DebugerHidden” attribute.

If you move from “EventView” mode to “CallView” mode in Intellitrace window, you can find the actual flow of sequence of the program.



Switch to IntelliTrace Events View

```
Calls for thread Main Thread (9288)
[System.Threading.ThreadHelper.ThreadStart()]
DebuggerHiddenDemo.Program.Main(string[] args = {string[0]})
DebuggerHiddenDemo.Program.Method1()
DebuggerHiddenDemo.Program.Method2()
    Function Entry: DebuggerHiddenDemo.Program.Method2
    Debugger : Breakpoint Hit: Method2, Program.cs line 32
```

Share this:

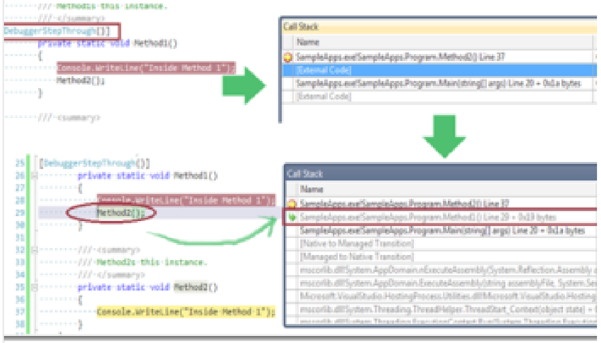
Tweet Share 0 Share Save WhatsApp Share

Like this:

Like

Be the first to like this.

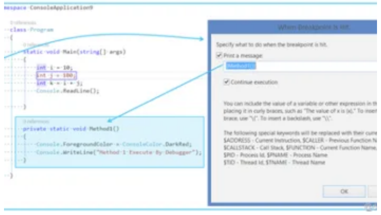
Related



Fav 3 : Three Favorite Tips from Daily .NET Tips – Series 1

#1 : Get list of all active session variables in ASP.NET This post talks about how you can get list of all active Session Variables in ASP.NET Application. The easiest way to get the details of session variable is using “Tracing” . If you enable

In "Fav 3"



Automatically calling a method when a breakpoint is hit

In "Debugging"

Using DebuggerStepThrough attributes to stepped over code during debugging
In "Tips"

Tips Debugging Visual Studio

Author: Abhijit Jana G+

Abhijit runs the Daily .NET Tips. He started this site with a vision to have a single knowledge base of .NET tips and tricks and share post that can quickly help any developers . He is a Former Microsoft ASP.NET MVP, CodeProject MVP, Mentor, Speaker, Author, Technology Evangelist and presently working as a .NET Consultant. He blogs at http://abhijitjana.net , you can follow him @AbhijitJana . He is the author of book Kinect for Windows SDK Programming Guide.

Related Posts

What is Azure Blob Rehydration ?

Quickly Move Between Methods in Visual Studio

Visual Studio Tooling for migrating NuGet Packages.config to PackageReference

Open your current project in Visual Studio Code directly from Visual Studio IDE

← Use “Obsolete” attributes to indicate Obsolete Methods

Using DebuggerStepThrough attributes to stepped over code during debugging →

3 thoughts on “Hide Methods from debugger Using DebuggerHidden attribute”

Pingback: SHRAVAN Weblog » 14 Useful .NET Debugging Tips & Tricks in Visual Studio

跟監
August 15, 2011

I loved as much as you will receive carried out right here. The sketch is attractive, your authored subject matter stylish. nonetheless, you command get got an nervousness over that you wish be delivering the following. unwell unquestionably come further formerly again since exactly the same nearly a lot often inside case you shield this hike. 5

Pingback: 10 Effective Debugging Tips for .NET Developer

Comments are closed.

Search

Email Subscription

Subscribe Daily .NET Tips for regular updates!

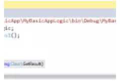
Join 8,841 other subscribers

Email Address

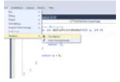
Subscribe

Popular Tips

Binding in Style Setters – SilverLight 5 Beta

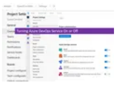

Calling current project methods from C# Interactive Window in Visual Studio 2015 ?

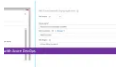
Use C# Interactive Window for your coding experiment in Visual Studio 2015


Exploring and Managing Unit Tests Using Test Explorer in Visual Studio

In Parameters in C# 7.2 – Read-only References

Recent Tips


Turning Azure DevOps Service On or Off


Connecting AWS with Azure DevOps

Setting up DevOps for Containerized Web Application using Azure DevOps Project

Code Samples Browser for Microsoft Platform Developers

Setting up DevOps Project for Azure IoT Edge application

Archives

Select Month ▼

Email Subscription

Subscribe Daily .NET Tips for regular updates!

Join 8,841 other subscribers

Email Address

Subscribe

Tags

.NET .NET 4.0 .NET Tips an Tricks Application Insights ASP.NET Azure Azure Data Factory Azure Portal Back To Basic C# csharp Debugging Developer Preview dotnet extensibility FAQ How To html5 JavaScript Kinect for Windows SDK Kinect for Windows SDK Tips Kinect Programming Tips LINQ MEF Productivity Tips Tips Tips & Tricks Tips and Tricks unit Test Universal Windows App Visual Studio visual Studio 2010 Visual Studio 2011 visual studio 2013 visual Studio 2013 Preview Visual Studio 2015 Visual Studio 2017 Visual Studio Editor VSTS Windows 10 Windows Phone Windows Universal App WPF Xamarin XAML

Contact

If you have any query, suggestions or corrections, sponsorship / advertisement inquiry feel free to send an email to contact@dailydotnettips.com

Advertisement | Contact Us | About