Kỹ năng Debug dành cho lập trình viên Winform

Kỹ năng debug là một kỹ năng nền tảng không thể thiếu của bất kỳ một lập trình viên nào. Debug không chỉ để tìm và sửa lỗi, mà còn giúp lập trình viên hiểu rõ hơn luồng của chương trình qua đó cải thiện kỹ năng code của mình thông qua các lỗi đã được sửa.

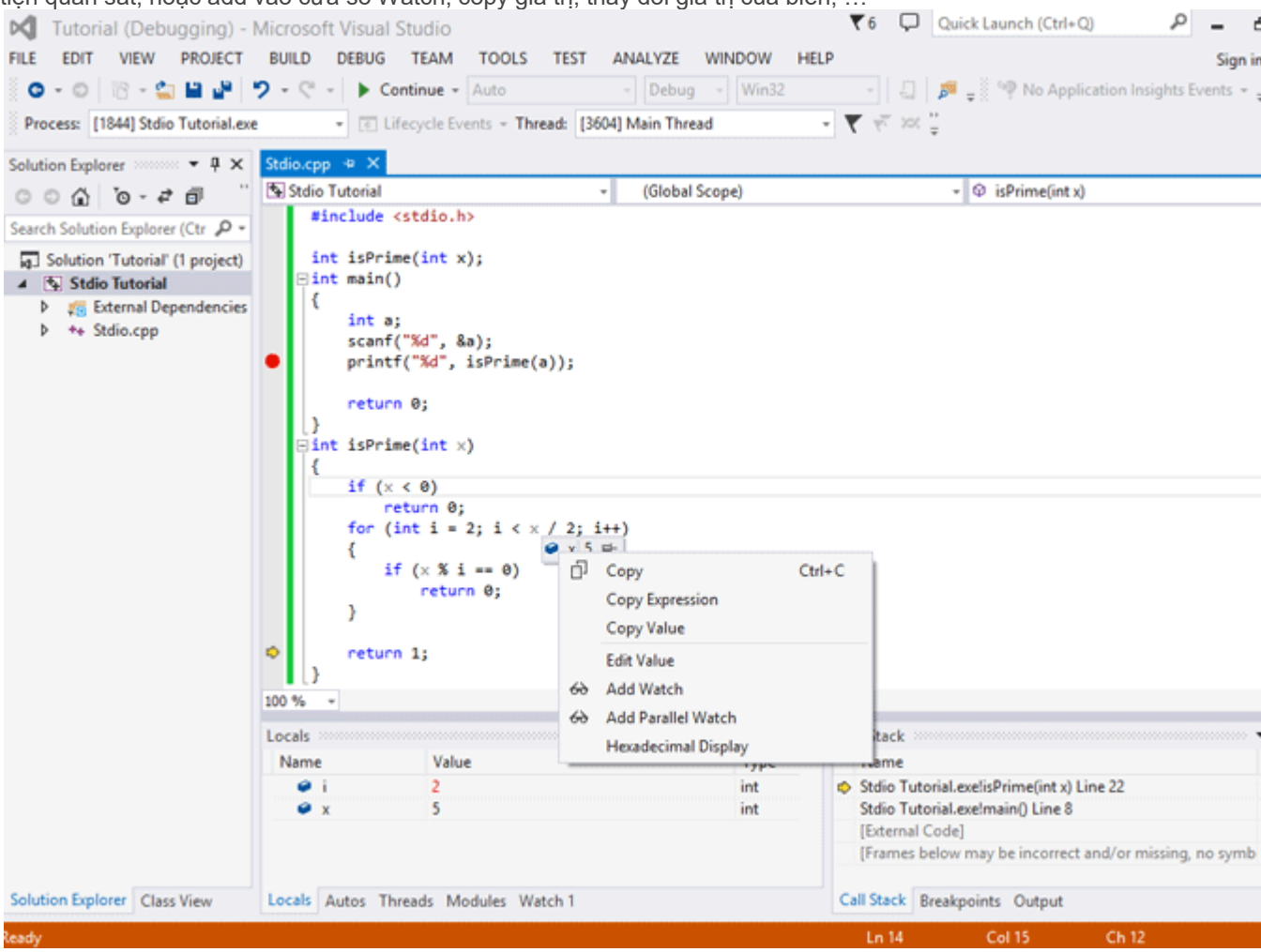
Muốn cải thiện kỹ năng Debug, trước tiên các bạn cần nắm được một số khái niệm cơ bản, sau đó sẽ là một vài Tips & Tricks khi bạn thực hiện debug trên các ứng dụng Winform.

1. Một số khái niệm cơ bản

- **Breakpoints**
  Breakpoints là vị trí mà chương trình sẽ dừng lại để Dev xem xét sự thay đổi của các biến qua từng dòng lệnh, từ đó phát hiện ra vị trí dòng code bị lỗi. Có thể đặt nhiều Breakpoint trong chương trình. Bạn có thể quản lý tất cả các Breakpoint của chương trình thông qua cửa sổ Breakpoints bằng cách nhấn Ctrl + Alt + B.

- **Data Tip**
  Khi di chuyển con trỏ chuột đến tên biến ở bất kì vị trí nào trong phạm vi cặp dấu { } (scope) hiện tại, giá trị của biến sẽ được hiển thị trên màn hình. Khi đó, các bạn có thể “ghim” biến đó lên màn hình để

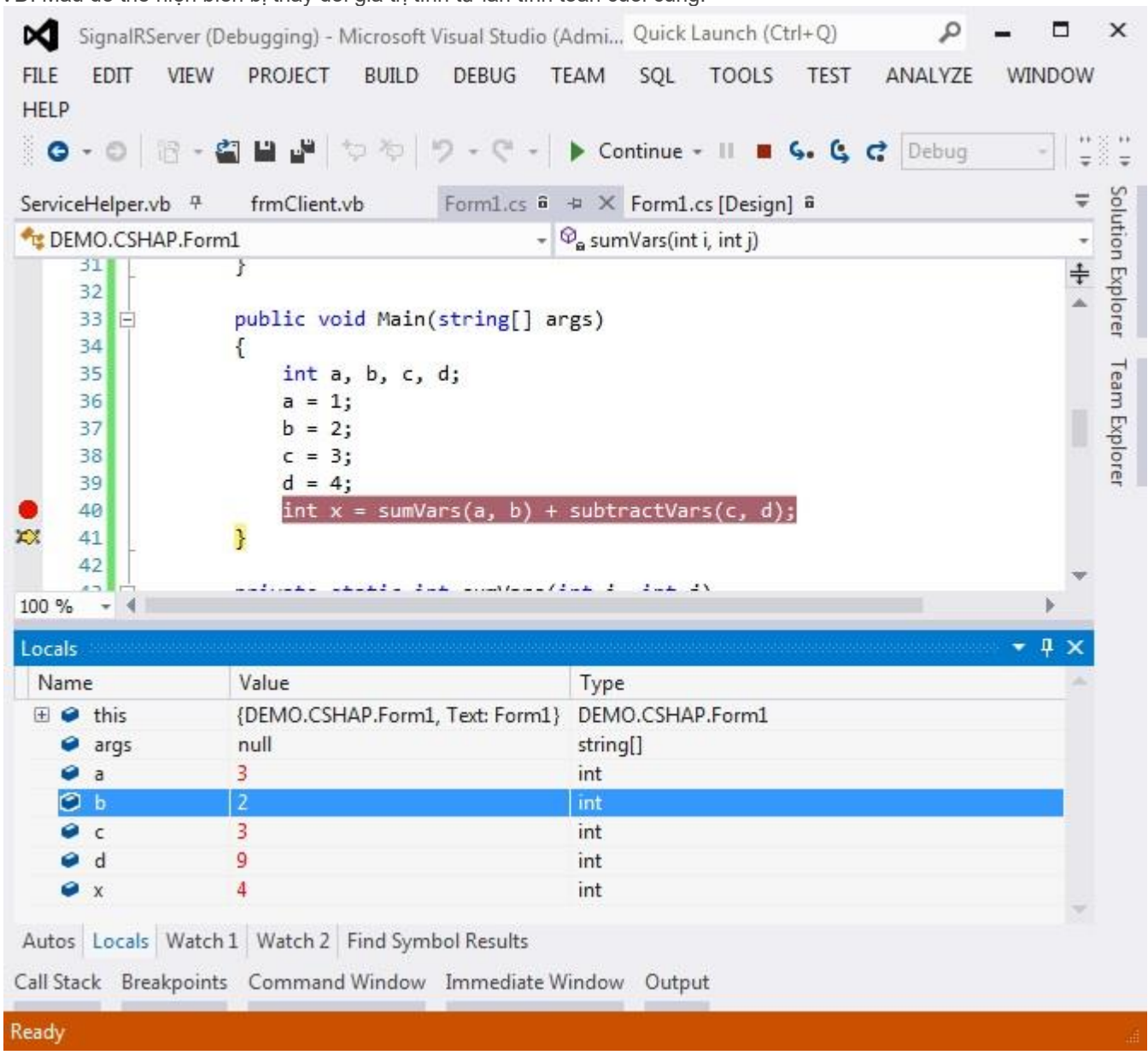
tiện quan sát, hoặc add vào cửa sổ Watch, copy giá trị, thay đổi giá trị của biến, …



- **Locals**

  Đây là cửa sổ sẽ hiển thị tất cả các biến có liên quan đến dòng code hiện tại một cách tự động. Các biến hiển thị ở đây sẽ được thay đổi qua từng dòng code. Ngoài ra, dựa vào màu sắc của các biến giúp ta phân biệt được những biến nào vừa thay đổi giá trị.

VD: Màu đỏ thể hiện biến bị thay đổi giá trị tính từ lần tính toán cuối cùng.



- **Autos**

Tương tự như Locals, cửa sổ Autos hiển thị các biến vừa được sử dụng trong dòng code hiện tại hoặc dòng trước. Visual Studio sẽ tự động nhận diện biến nào không còn cần thiết và loại bỏ ra khỏi cửa sổ Autos.

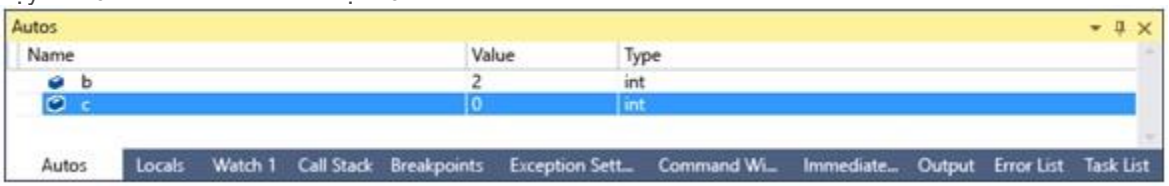
VD:

```
1 public static void Main()
```

```
2 {
3     int a, b, c, d;
4     a = 1;
5     b = 2;
6     c = 3;
7     d = 4;
8 }
```

Nếu bạn đặt debug vào dòng c = 3;

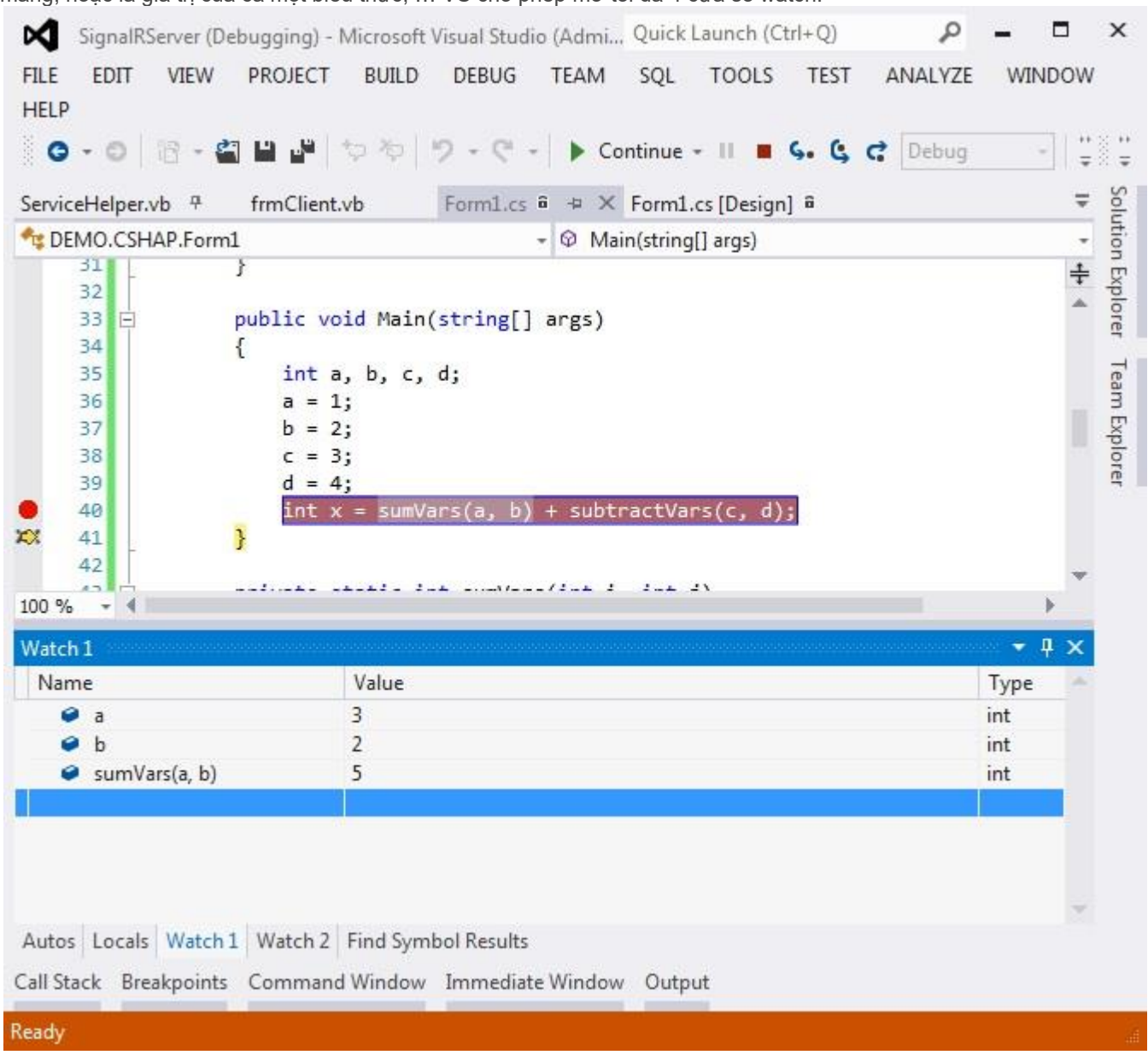
Vậy thì cửa sổ Autos sẽ hiển thị như sau:



- **Watch**

  VS không thể biết được tất cả những gì một Dev cần để tìm ra lỗi, vì vậy VS cung cấp các cửa sổ cho phép Dev có thể xem các biến tùy theo yêu cầu của Dev. Có thể là giá trị của phần tử thứ 4 của một

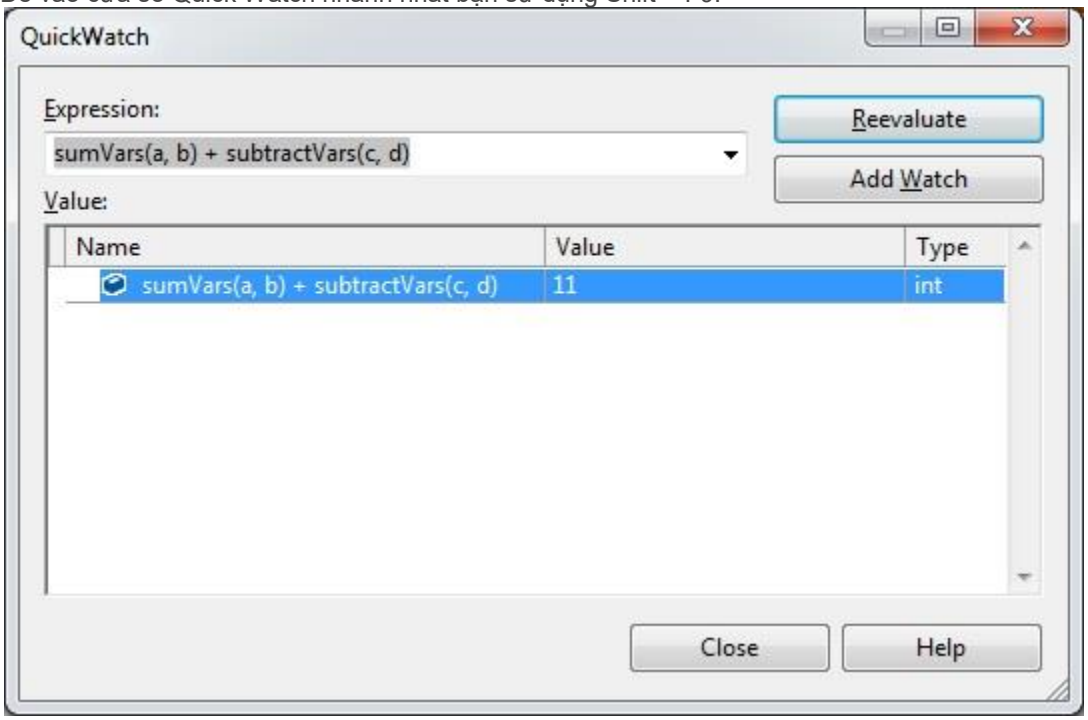
mảng, hoặc là giá trị của cả một biểu thức, … VS cho phép mở tối đa 4 cửa sổ watch.



- **Quick watch**

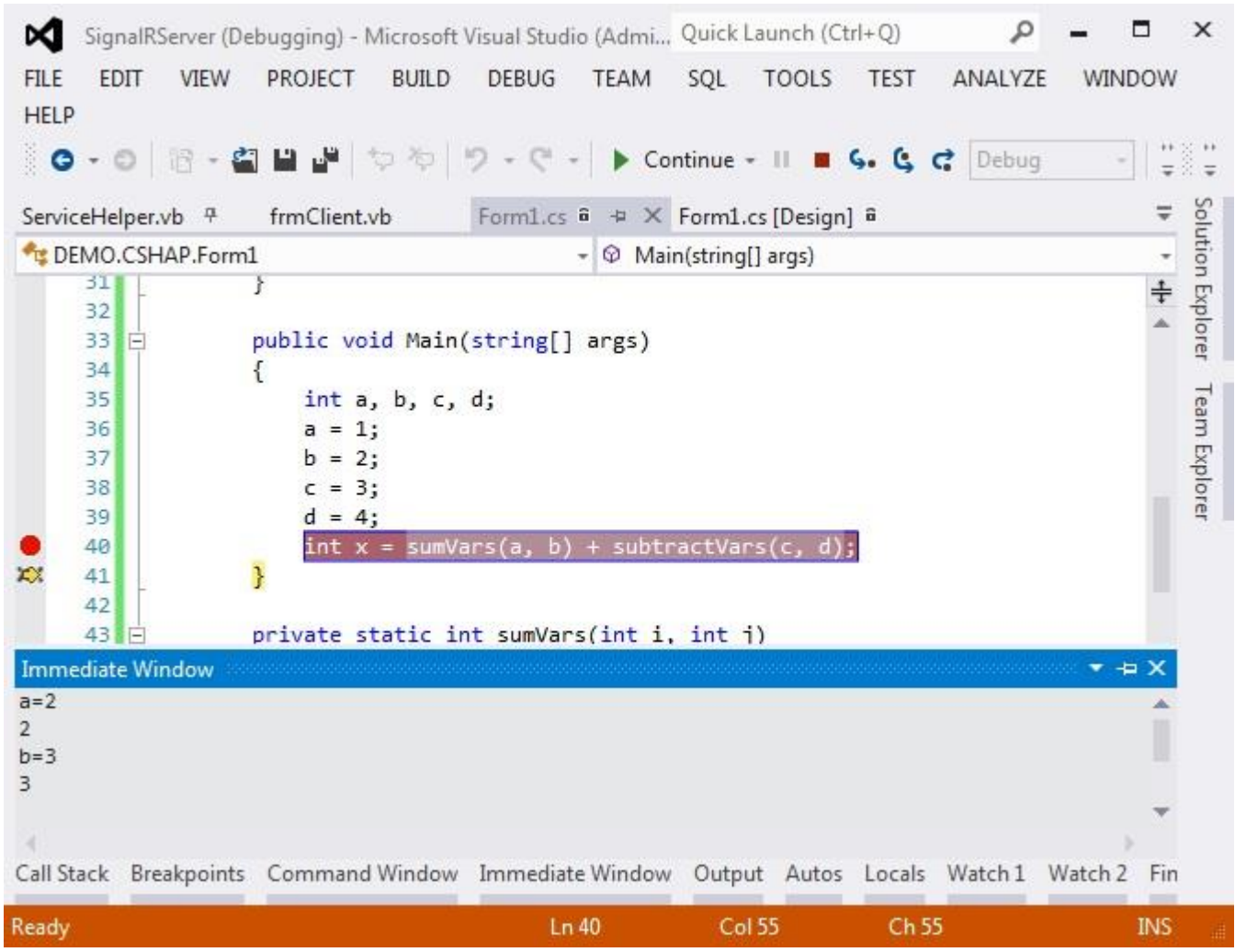
  Cửa sổ này tương tự như cửa sổ Watch, có 1 điểm khác là cửa sổ Quick Watch là dialog, vì vậy bạn không thể tiếp tục debug nếu như chưa đóng cửa sổ này lại. Nếu muốn theo dõi tiếp một biến hoặc một biểu thức nào đó, bạn hoàn toàn có thể add vào cửa sổ Watch rồi đóng lại và tiếp tục quá trình debug.

Để vào cửa sổ Quick Watch nhanh nhất bạn sử dụng Shift + F9.

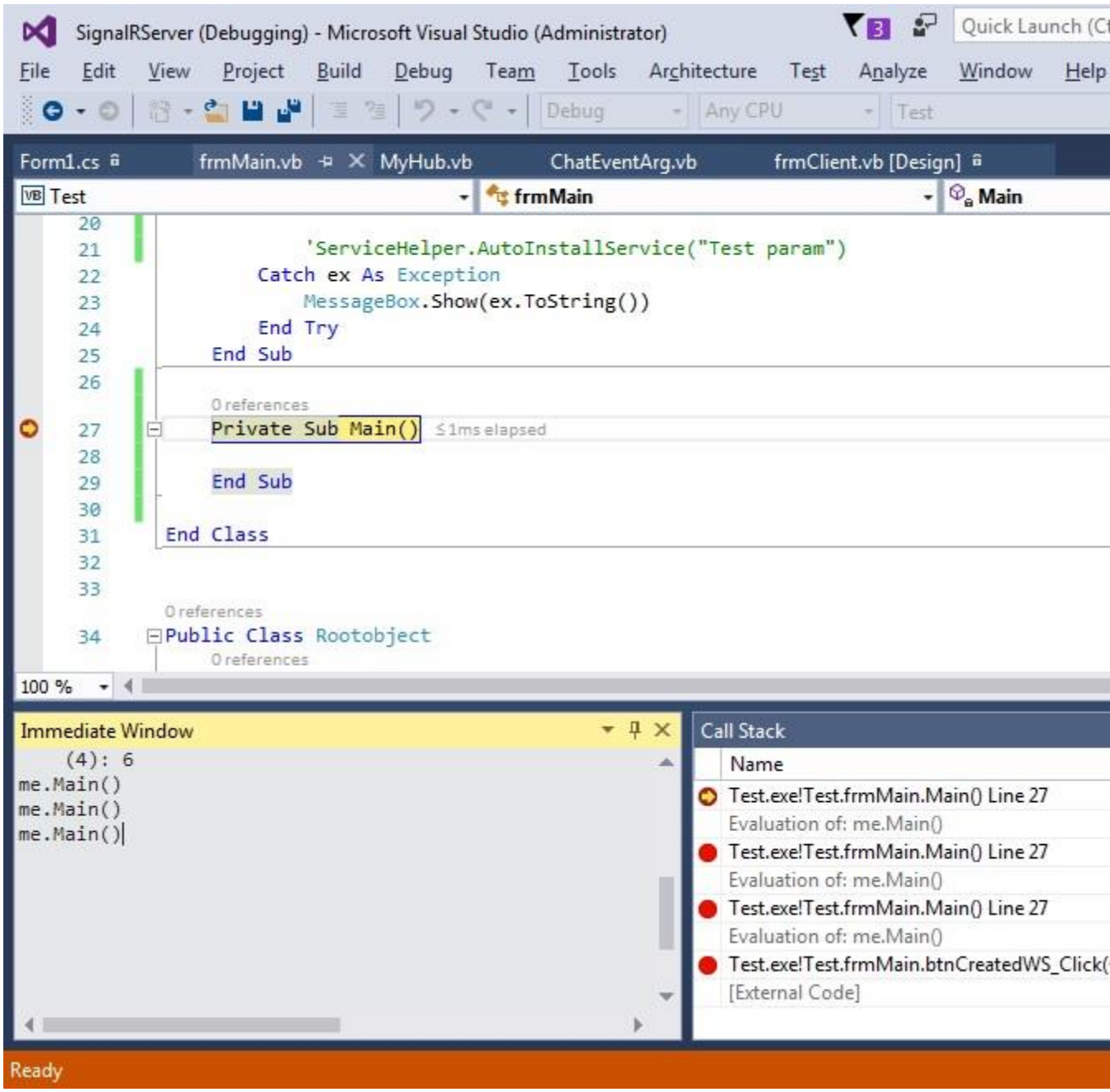


- **Immediate**

  Cửa sổ này cho phép bạn lấy ra giá trị của các biến biểu thức, gán lại giá trị nếu muốn, gọi hàm, … Điểm tiện ích của cửa sổ này là nó sẽ lưu lại toàn bộ các lệnh của bạn. Bạn hoàn toàn có thể dùng Up/Down arrow để có thể di chuyển đến câu lệnh mong muốn (giống với màn hình cmd của window) hoặc có thể dùng chuột để copy câu lệnh xuống.



Để xóa hết các dòng lệnh bạn có thể gõ “>cls. Để run các command bạn phải có ký tự >. Trong cửa sổ này, dấu ? sẽ tương đương với lệnh **Debug.Print**.



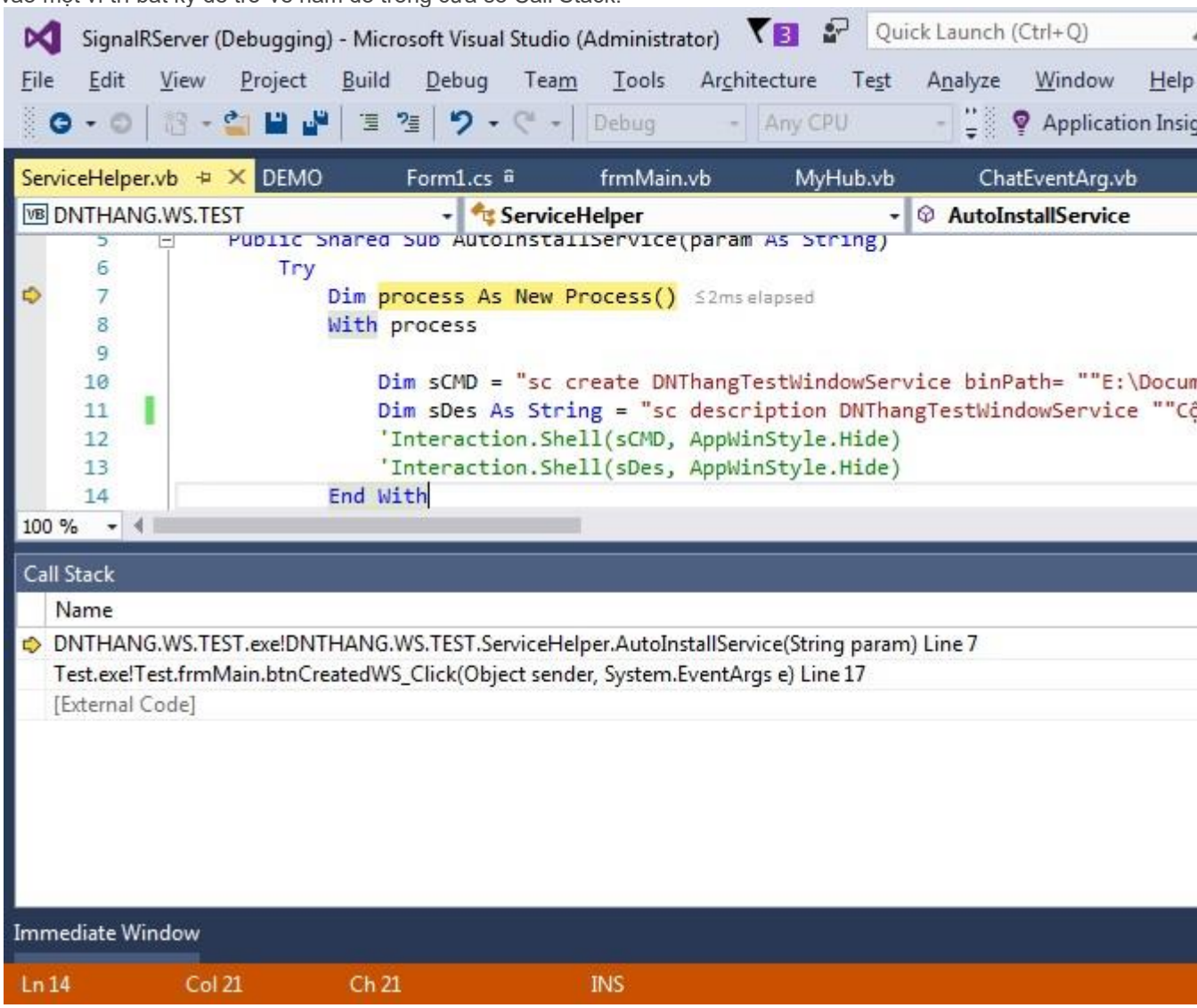
Như bạn thấy, tôi đã đặt breakpoint vào hàm Main(), sau đó ở trong cửa sổ Immediate mình gọi vào phương thức đó 3 lần và cửa sổ Call Stack xuất hiện 3 breakpoint khác nhau tương ứng với từng lần gọi phương thức. Việc debug này hoàn toàn độc lập với chương trình đang chạy hiện thời. Bạn có thể debug như bình thường hoặc có thể nhấn F5 hoặc Continue để tiếp tục.

Mặt khác, bạn có thể dùng cửa sổ này để log lại thông tin từ hàm**Debug.WriteLine()** hoặc **Debug.Print()**

- **Call Stacks**

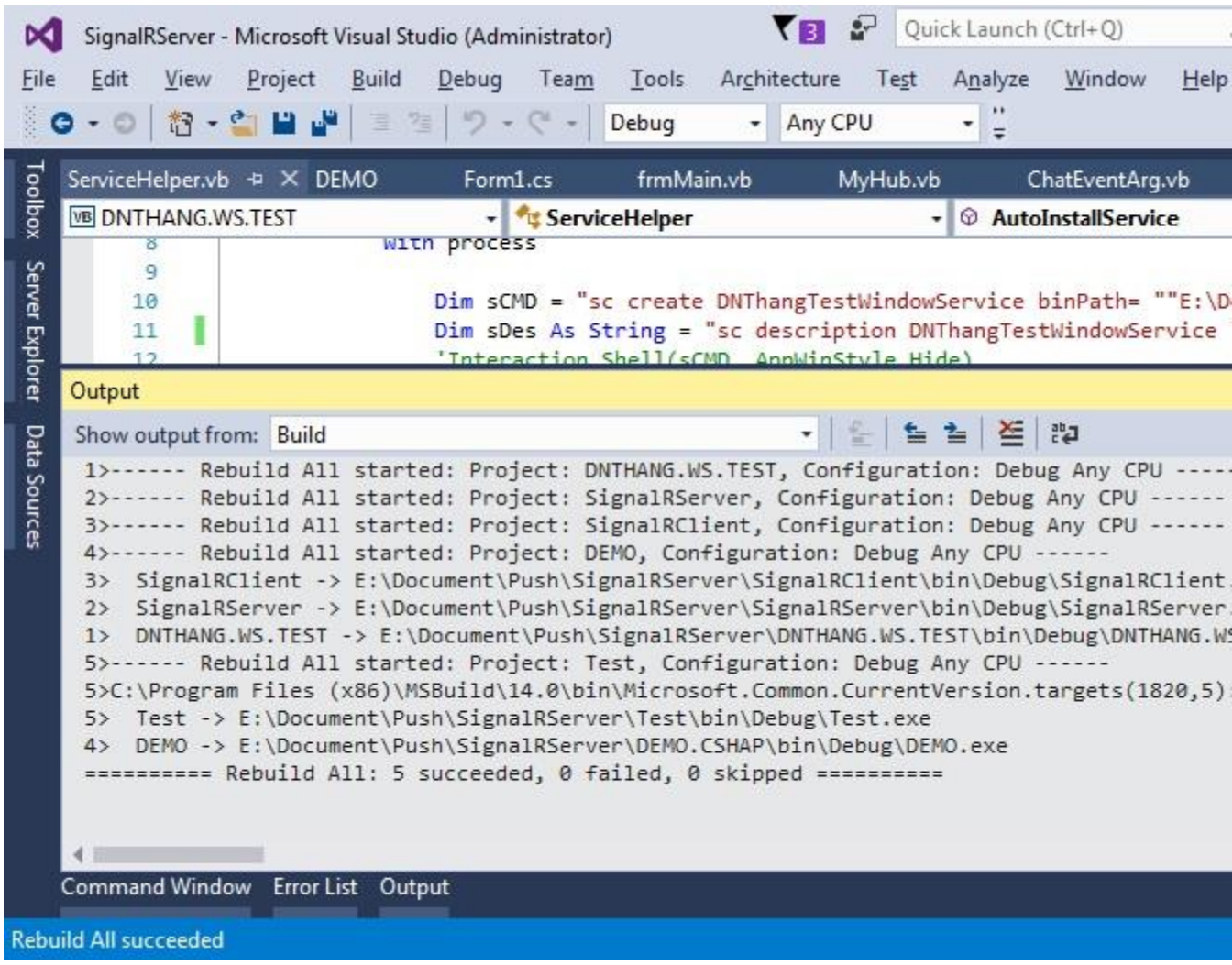
  Bạn đã bao giờ gặp tình huống các Function này gọi Function kia, mà cụ thể là bài toán đệ quy. Có quá nhiều hàm gọi lồng vào nhau, lúc này cửa sổ Call Stack chính là cánh tay đắc lực cho bạn vào lúc

này. Cửa sổ này sẽ lưu lại thứ tự tất cả các lời gọi hàm trong ngăn xếp. Bạn hoàn toàn có thể click vào một ví trí bất kỳ để trở về hàm đó trong cửa sổ Call Stack.



- **Output**

  Đã khi nào solution của bạn khi build ra có rất nhiều lỗi mà bạn chưa biết là sẽ bắt đầu từ đâu. Khi gặp tình huống như thế này thì bạn nên nghĩ ngay đến cửa sổ Output, cửa sổ này sẽ lưu lại toàn bộ các thông tin của quá trình build ứng dụng, khi đó bạn chỉ cần tìm ra dòng đầu tiên gặp lỗi trong cửa sổ Output để sửa, nếu vẫn còn lỗi thì bạn tiếp tục làm như vậy.



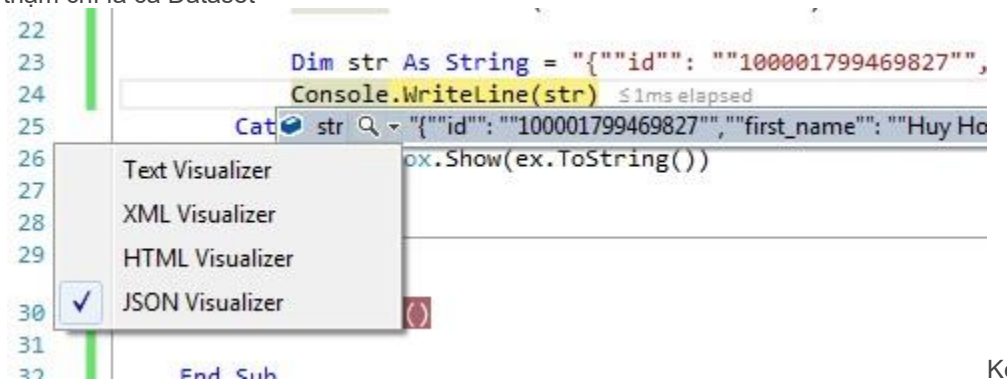
Nhưng mà không lẽ cửa sổ này chỉ có tác dụng như thế thôi sao? Để tiện cho việc Debug, bạn hoàn toàn có thể dùng cửa sổ này để log thông tin mà không cần phải xử lý lưu ra file hoặc dùng một thư viện Log nào đó (nếu chỉ là cần để dò lỗi trong quá trình phát triển) bằng cách sử dụng Console.WriteLine(). Bạn có thể chuyển các message của class Debug vào cửa sổ Output bằng cách sử dụng câu lệnh “Debug.Listeners.Add(New TextWriterTraceListener(Console.Out))”). Mặt khác cửa sổ này được sử dụng để hiển thị thông báo của nhiều tool, chức năng của VS nữa (VD: diagnostic…)

- **Visualizer**

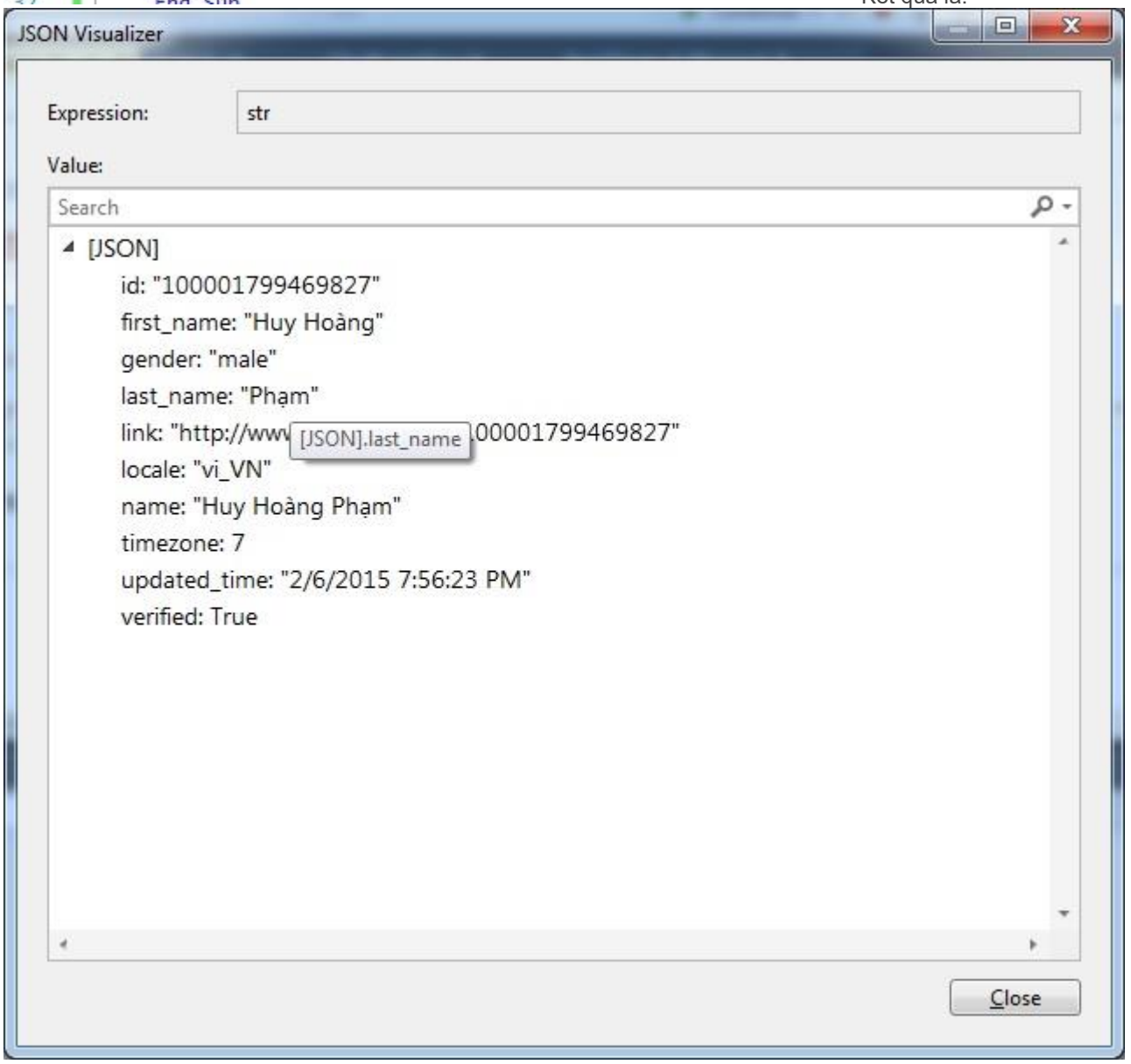
Thôngthường, nếu bạn muốn debug để xem các giá trị String, Integer, Guid, … thì bạn hoàn toàn có thể sử dụng các cửa sổ Autos/Locals/Watch/Imediate/… Thế nhưng với tình huống này chẳng hạn:

Nếu nhìn đoạn text trên thì bạn cũng khó hình dung. Thật may là trên VS lại hỗ trợ cho chúng ta một món gọi là Visualizer, nó giúp hiển thị kết quả trả về một cách trực quan hơn như html, json, xml hay

thậm chí là cả Dataset



Kết quả là:



2. Một số phím tắt thường dùng trong quá trình Debug

Debug	
F5	Chạy ứng dụng attach với process
Ctrl + F5	Chạy ứng dụng không attach với process (Không dùng công cụ debug)
Ctrl+Shift+F5	Khởi động lại ứng dụng
Ctrl+Alt+P	Attach solution hiện tại vào process
Ctrl+Alt+B	Hiển thị cửa sổ Breakpoints
Ctrl+Alt+C	Hiển thị cửa sổ Call Stack
Ctrl+Alt+E	Hiển thị cửa sổ Exceptions
Ctrl+Alt+I	Hiển thị cửa sổ Immediate
Ctrl+Alt+W, 1	Hiển thị cửa sổ Watch 1

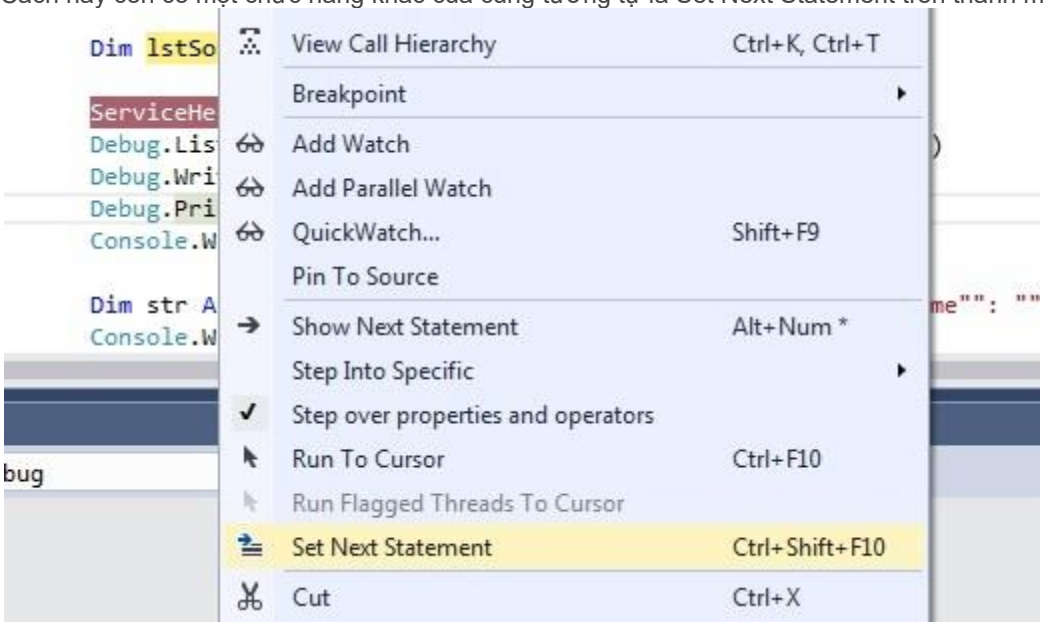
Shift+F9 Hoặc Ctrl+Alt+Q	Show cửa sổ QuickWatch
F9	Đặt breakpoint
F10	Chạy lần lượt các câu lệnh, không vào trong function con
F11	Debug qua từng câu lệnh, nếu gặp function con thì thực hiện debug sâu cả vào function (nếu được)
Ctrl+F10	Chạy ứng dụng đến vị trí của con trỏ hiện tại
Ctrl+F9	Enable/Disable Breakpoint
Ctrl+Shift+F9	Xóa các breakpoint
Shift+F11	Lướt qua hàm con hiện tại để trở về hàm trước hoặc nhảy đến Breakpoint kế tiếp

3. Một số Tips & Tricks khi Debug với Visual Studio

- **Set Next Statement**

  Trong quá trình Debug có khi nào bạn sơ ý nhấn F10 qua đoạn code mà mình cần debug hoặc là sau khi debug qua thì mình mới thấy đoạn code vừa qua có dấu hiệu khả nghi chẳng hạn. Lúc này mà chạy lại từ đầu thì sẽ thật là mất công, đặc biệt là việc debug vào đến trong này lại chẳng dễ dàng gì. Lúc này hoàn toàn bạn có thể sử dụng tới cách: Dùng chuột để nắm và kéo mũi tên vàng ở bên trái và thả vào đoạn code nào mà bạn muốn debug tiếp.

  Cách này còn có một chức năng khác của cũng tương tự là Set Next Statement trên thanh menu.



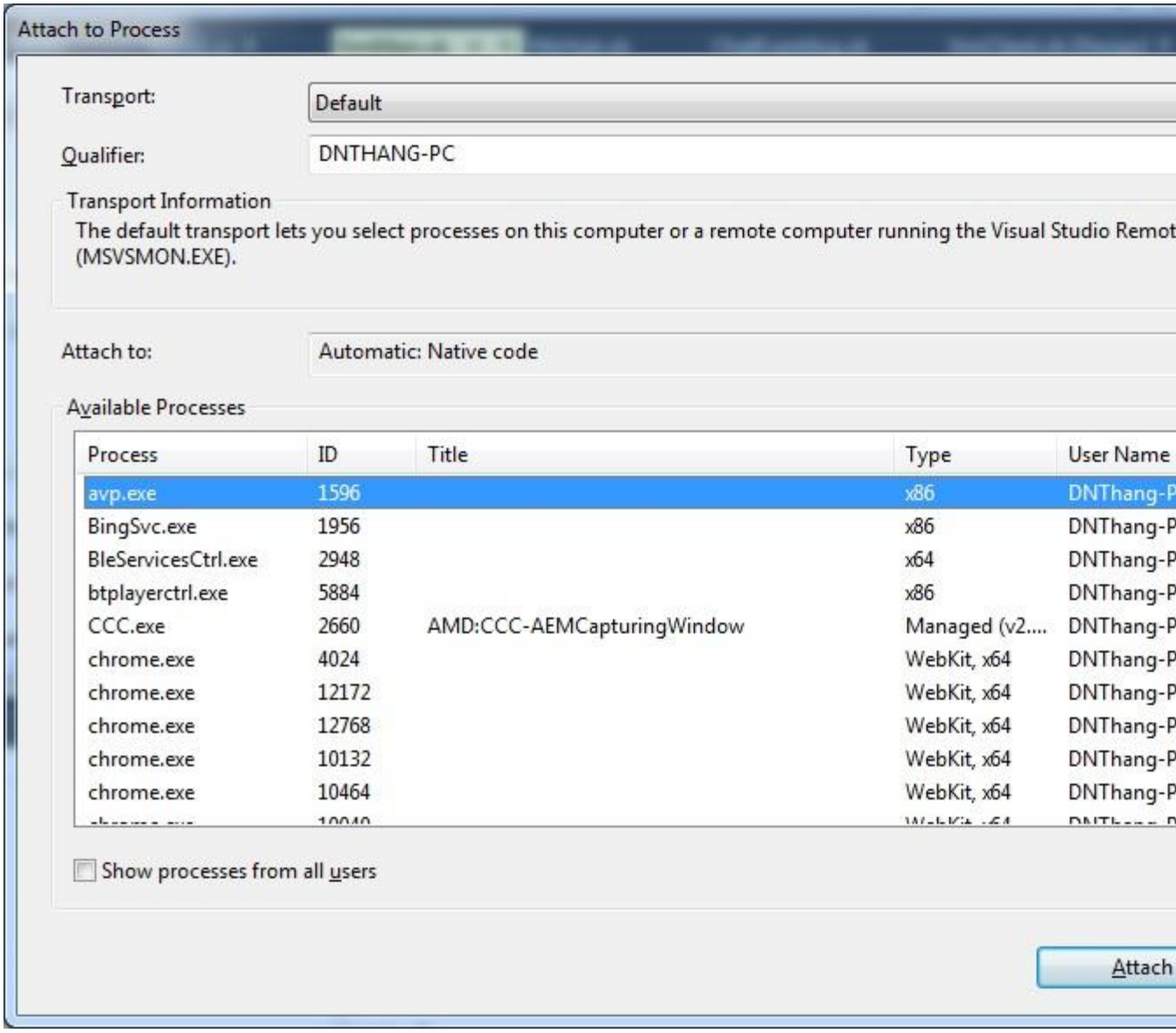
  Chức năng này khác một điều là chỉ hỗ trợ chạy đến đoạn code tiếp theo mà không cho chạy ngược lại, và bạn chú ý là nếu dùng chức năng này có thể làm sai logic code của bạn nên cần cân nhắc trước khi sử dụng. Thông thường mình hay sử dụng chuột để nắm và kéo mũi tên vàng ở bên trái để debug lại hơn.

- **Edit value**

  Khi debug chắc chắn tôi và các bạn đều có nhu cầu muốn sửa lại giá trị của tham số, biến… Không cần thiết bạn phải sửa code để set giá trị cần thiết hoặc chạy lại chương trình để nhập lại, … Bạn hoàn toàn có thể khai thác bất kỳ các tính năng nào của VS để sửa lại giá trị: Immediate, Watch, QuickWatch, DataTip, Local,…

- **Attach process**

  Với nhiều dự án lớn, việc build ra mất rất nhiều thời gian, nên ta hay dùng cách là chỉ chạy file build ra hoặc chạy Ctrl+F5, nhưng mà nếu có lỗi mà mình lại phải chạy lại để debug thì khá là bất tiện. Giải pháp hoàn hảo cho tình huống này là bạn có thể attach code trên VS vào ứng dụng đang chạy và bạn hoàn toàn có thể debug bình thường. Để thực hiện bạn nhấn tổ hợp phím Ctrl+Alt+P, VS sẽ show lên cửa sổ Attach process, việc của bạn lúc này là chỉ việc tìm đúng proces của ứng dụng đang chạy và nhấn Attach.



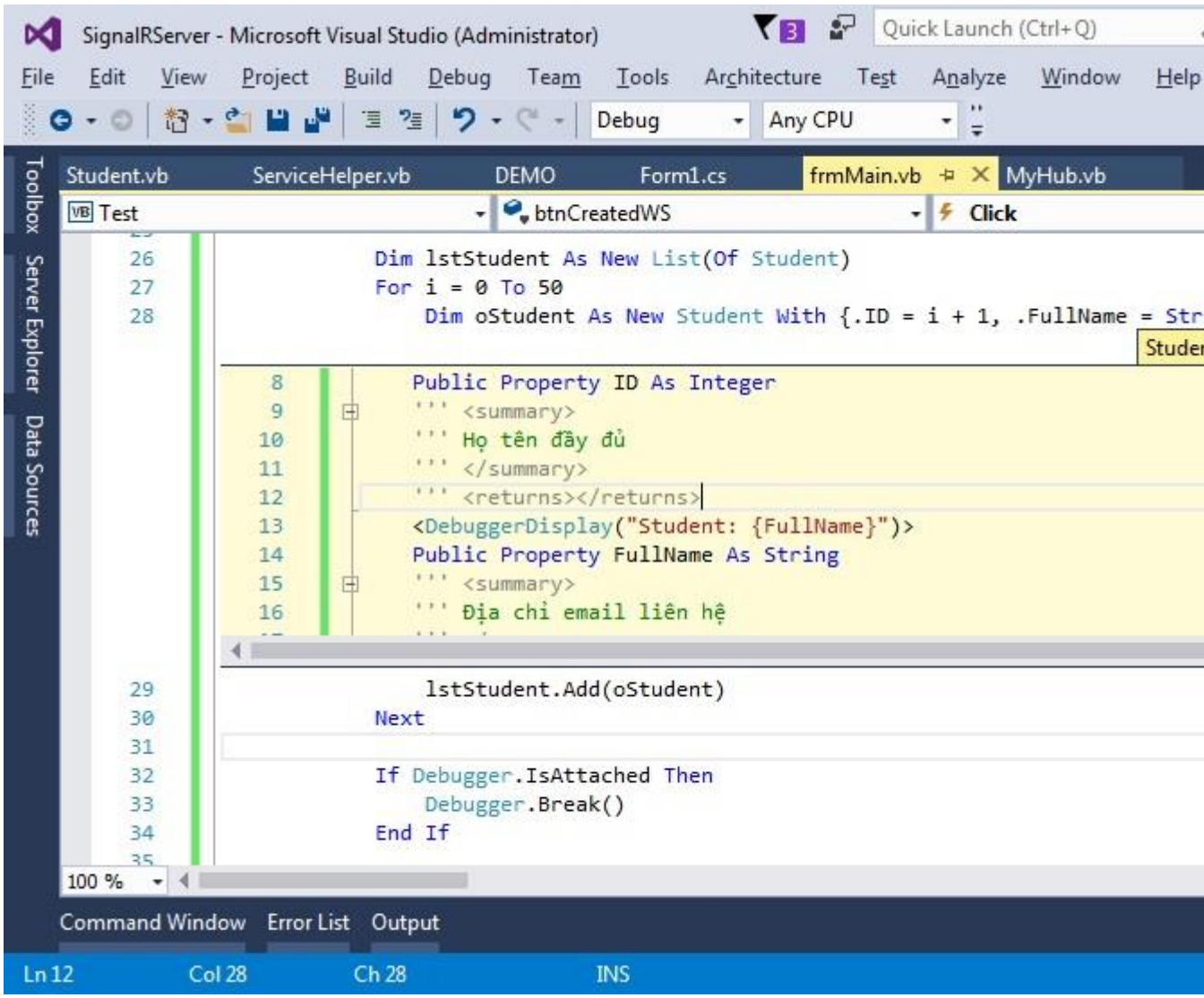
Tính năng này còn hiệu quả hơn nữa khi bạn rơi vào tình huống chương trình lại chỉ bị lỗi trên một máy nào đó mà không phải máy dev, có thể là máy QC hoặc máy trắng để test hoặc là khi bạn làm web mà muốn kiểm tra lỗi trên máy chủ hosting, khi đó bạn có thể sử dụng Remote Debug kết hợp với Attach Process.

- **DebuggerDisplay Attribute và Peek Definition**

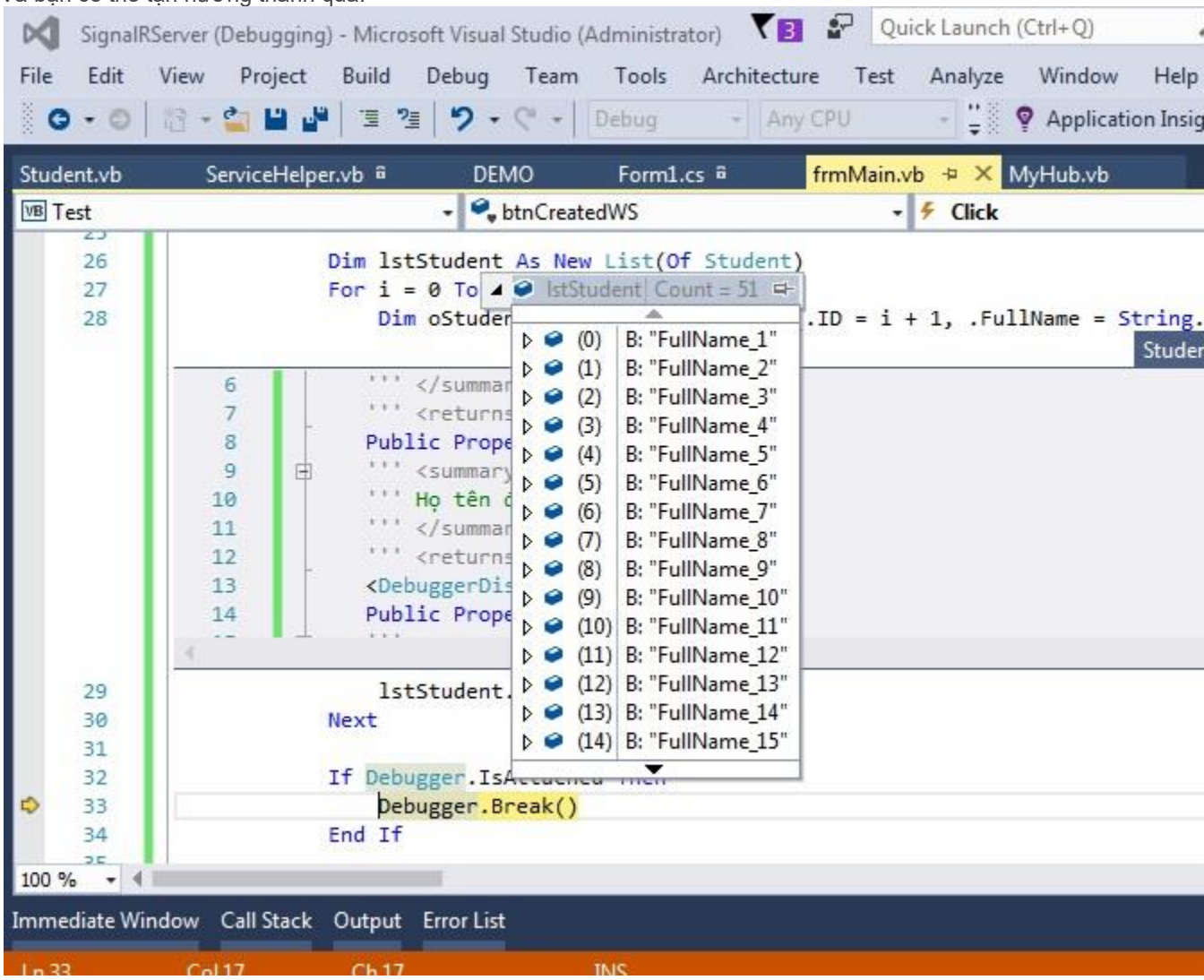
  Khi làm việc với Entity, chắc hẳn bạn cũng sẽ rơi vào tình huống mà lấy ra cả danh sách các Entity (List(Of Student)), mà đối tượng này có rất nhiều thuộc tính bên trong nhưng mà mình lại chỉ cần quan tâm đến một thuộc tính nào đó. Thật phiền phức nếu cứ mỗi Item mình lại phải bấm vào mũi tên rồi xổ ra danh sách các thuộc tính để xem giá trị. Hoặc không thì bạn dùng các cửa sổ Watch/Local thì cũng chỉ xem được 1 giá trị. Nếu rơi vào tình huống này bạn có thể dùng giải pháp sau:
  VD: Mình có đối tượng Student và được khai báo như sau:

```
1  Public Class Student
2  #Region "Properties"
3      ''' <summary>
4      '''
5      ''' </summary>
6      ''' <returns></returns>
7      <Key()>
8      Public Property ID As Integer
9      ''' <summary>
10     ''' ?? tên đầy đủ
11     ''' </summary>
12     ''' <returns></returns>
13     Public Property FullName As String
14     ''' <summary>
15     ''' Địa ?? email liên ??
16     ''' </summary>
17     ''' <returns></returns>
18     Public Property Email As String
19 #End Region
20 End Class
```

Trong VS đã hỗ trợ DebuggerDisplay Attribute cho phép bạn có thể tùy biến giá trị hiển thị ra khi debug. Mặt khác để sửa nhanh đối tượng Student từ class hoặc form đang debug bạn có thể sử dụng ngay tính năng Peek Definition (Trỏ vào class Student trong form đang Debug và nhấn tổ hợp phím Alt + F12) dùng để hiển thị ra một popup nhỏ tương tự như Go to Definition nhưng bạn lại không phải chuyển giữa 2 cửa sổ làm việc.


SignalRServer - Microsoft Visual Studio (Administrator)
Quick Launch (Ctrl+Q)
File Edit View Project Build Debug Team Tools Architecture Test Analyze Window Help
Debug
Any CPU
Toolbox
Server Explorer
Data Sources
Student.vb
ServiceHelper.vb
DEMO
Form1.cs
frmMain.vb
MyHub.vb
Test
btnCreatedWS
Click
Dim lstStudent As New List(Of Student)
For i = 0 To 50
Dim oStudent As New Student With {.ID = i + 1, .FullName = Str
Public Property ID As Integer
''' <summary>
''' Họ tên đầy đủ
''' </summary>
''' <returns></returns>
<DebuggerDisplay("Student: {FullName}")>
Public Property FullName As String
''' <summary>
''' Địa chỉ email liên hệ
lstStudent.Add(oStudent)
Next
If Debugger.IsAttached Then
Debugger.Break()
End If
100 %
Command Window
Error List
Output
Ln 12
Col 28
Ch 28
INS

Và bạn có thể tận hưởng thành quả:

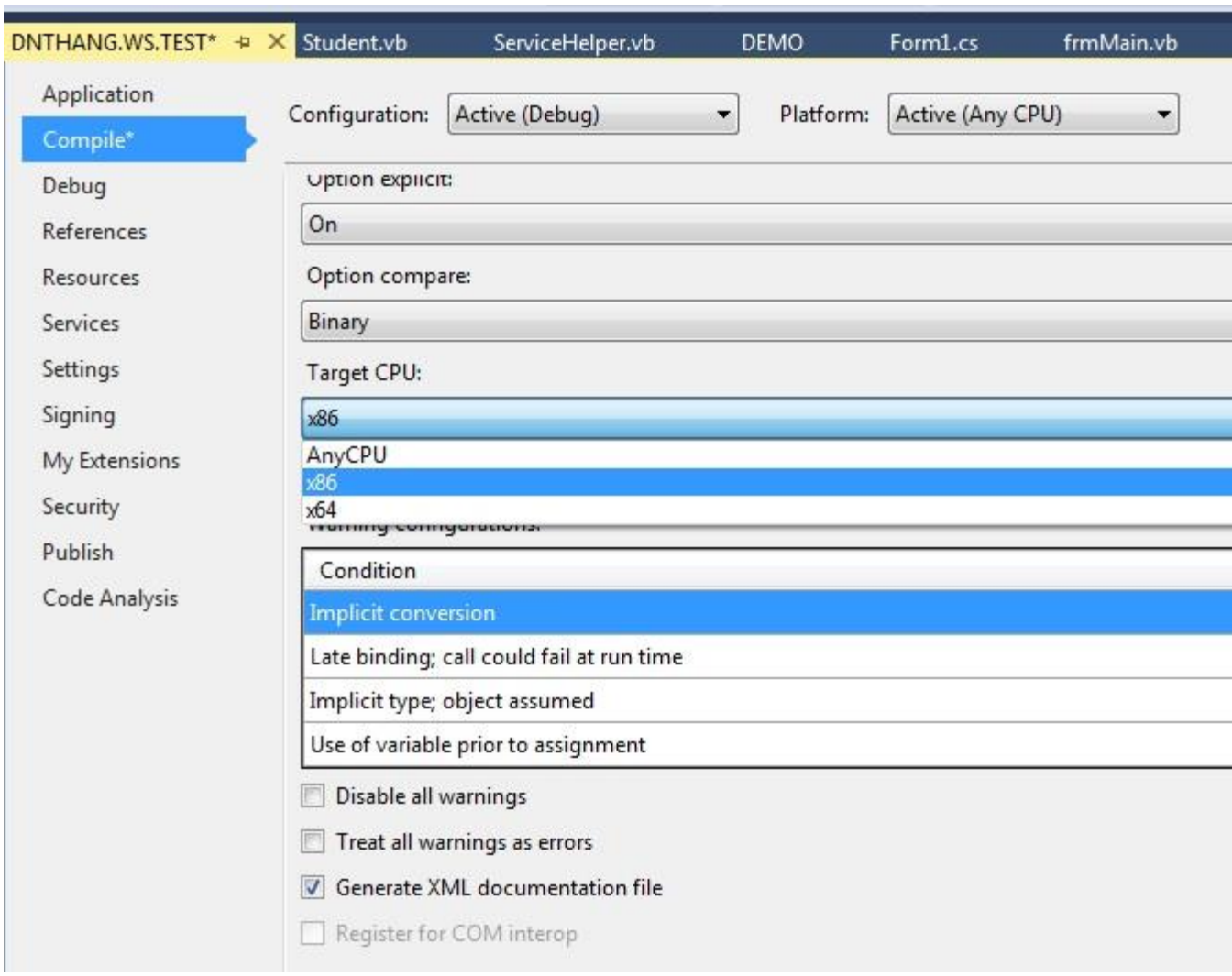


- **EditValue và Continue**

  Công việc Debug đòi hỏi phải rất tỉ mỉ, cẩn thận nên mất nhiều thời gian, nếu trong quá trình debug bạn đã phát hiện ra đoạn code cần sửa, bạn chỉ việc sửa lỗi của đoạn code đó và tiếp tục debug mà không cần restart lại quá trình Debug. Nếu bạn là một lập trình viên Winform mà không khai thác được lợi thế này thì quả là lãng phí.

  Tuy nhiên tính năng này có giới hạn.

  - Một là nó không đáp ứng cho các đoạn mã viết cho HĐH 64 bit, để thay đổi bạn chuột phải vào project chọn Properties\Tab Compile\Target CPU và chuyển về x86.

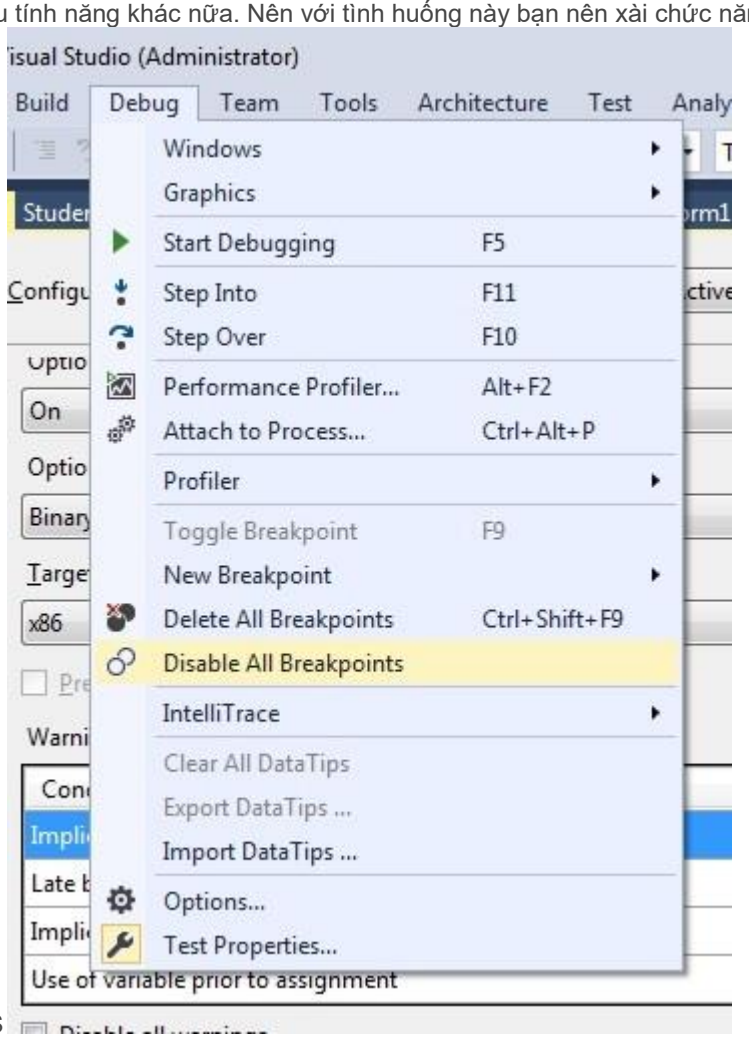


- Hai là trong quá trình bạn sửa source code, nếu bạn thay đổi khai báo hàm, tham số trong hàm, thêm mới hàm, hoặc thay đổi linq, … thì có thể sẽ phải restart lại quá trình Debug

- **Disable All Breakpoints**

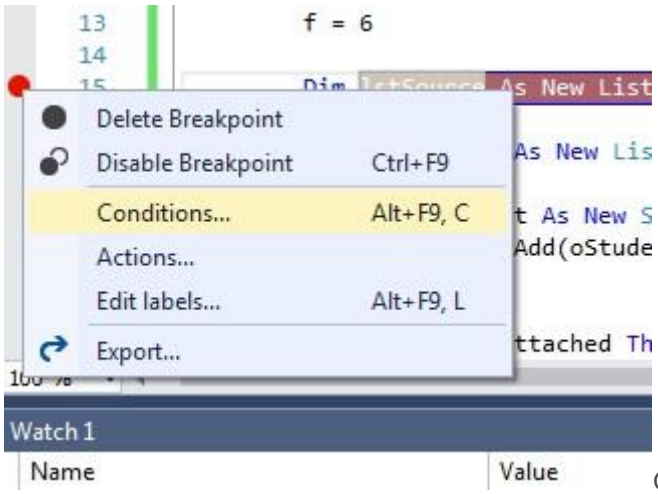
  Khả năng có khá nhiều bạn sẽ phải sang hỗ trợ cho các dev khác, hoặc là tình huống dev lead đi review và hỗ trợ, quả là hơi vô ý khi bạn bỏ hết các Breakpoints mà người được hỗ trợ đã đặt trước đó. Có thể người đó phải debug rất lâu mới đặt được các Breakpoint vào những điểm then chốt, có

thể là của của nhiều tính năng khác nữa. Nên với tình huống này bạn nên xài chức năng Disable All Breakpoints của VS



- **Breakpoint Setting**

  Hẳn là bạn đã quá quen thuộc với việc cứ phải nhấn F10 liên tục để vòng lặp chạy đến được với Item mà mình cần debug, hoặc là nếu bạn lập trình đa luồng thì sẽ hiểu sự khó khăn của của việc tìm ra được thread nào đang có vấn đề. May mắn là VS hỗ trợ cho lập trình viên một tính năng rất power đó là Breakpoint Settings. Tính năng này cho phép bạn tùy chỉnh các thiết lập của Breakpoint như điều kiện (Conditions), hành động (Actions). Để sử dụng, bạn chỉ cần đưa chuột vào Breakpoint cần thiết



lập rồi chuột phải. Ở mục Conditions bạn có thể lựa chọn:

- **Conditional Expression**: kiểm tra xem nếu điều kiện đúng (Is true) hoặc giá trị thay đổi (Has changed) thì tiến hành việc debug.
- **Hit Count**: kiểm tra số lần chạy tới dòng breakpoint. Bạn sẽ không còn gặp cảnh bấm F10 cả trăm lần chỉ để vòng lặp chạy đến item thứ 20 của mảng.
- **Filter**: cho phép kiểm tra các trường ThreadId, ThreadName, ProcessId, ProcessName, MachineName. Điều này sẽ cực kì hiệu quả nếu bạn đang lập trình đa luồngỞ mục **Actions** cho phép ta log message ra ngoài Output Window, ngoài ra Continue execution cho phép Debugger không dừng chương trình của mình lại (Tracepoints – thể hiện bằng nút breakpoint hình thoi, thay vì hình tròn). Đây là 1 tính năng rất hay vì nó cho phép ta log các biến ra ngoài cửa sổ Output Window mà mình vẫn có thể tiếp tục thực hiện các thao tác khác.

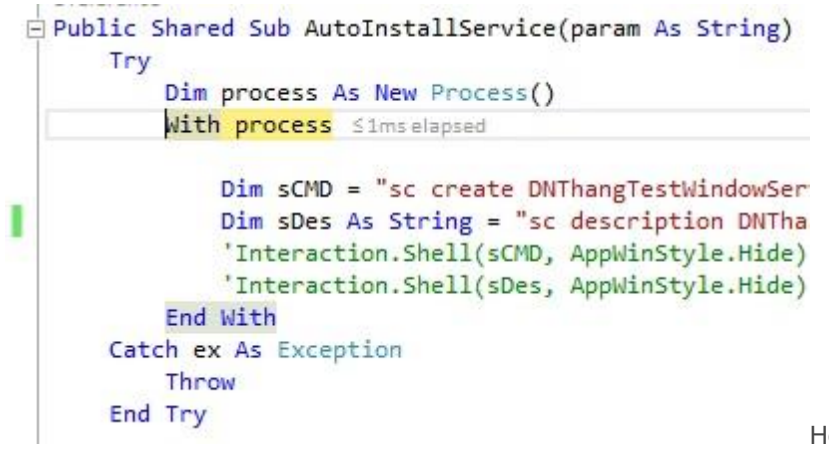
- Sử dụng Debug, Debugger

  Class này chứa rất nhiều hàm cho phép bạn sử dụng trong quá trình Debug, đặc biệt là VS sẽ không compile vào mã nguồn đóng gói khi build release. Các hàm hay dùng:

  - **Debug.Assert**: Kiểm tra điều kiện, nếu false thì sẽ hiển thị thông báo
  - **Debugger.Break**: Tương đương với việc đặt 1 Breakpoint
  - **Debug.Print, Debug.Write, Debug.WriteLine**: In message ra Immediate hoặc Output
  - **Debug.Listeners**: Tùy chọn đầu ra của message (Immediate, Output)
  - **Debugger.Launch**: Chạy và attach chương trình vào process, nếu như bạn phát triển ứng dụng có window service mà bạn cần phải debug vào hàm khi WindowService chạy thì lúc này Debugger.Launch là một cánh tay đắc lực cho bạn

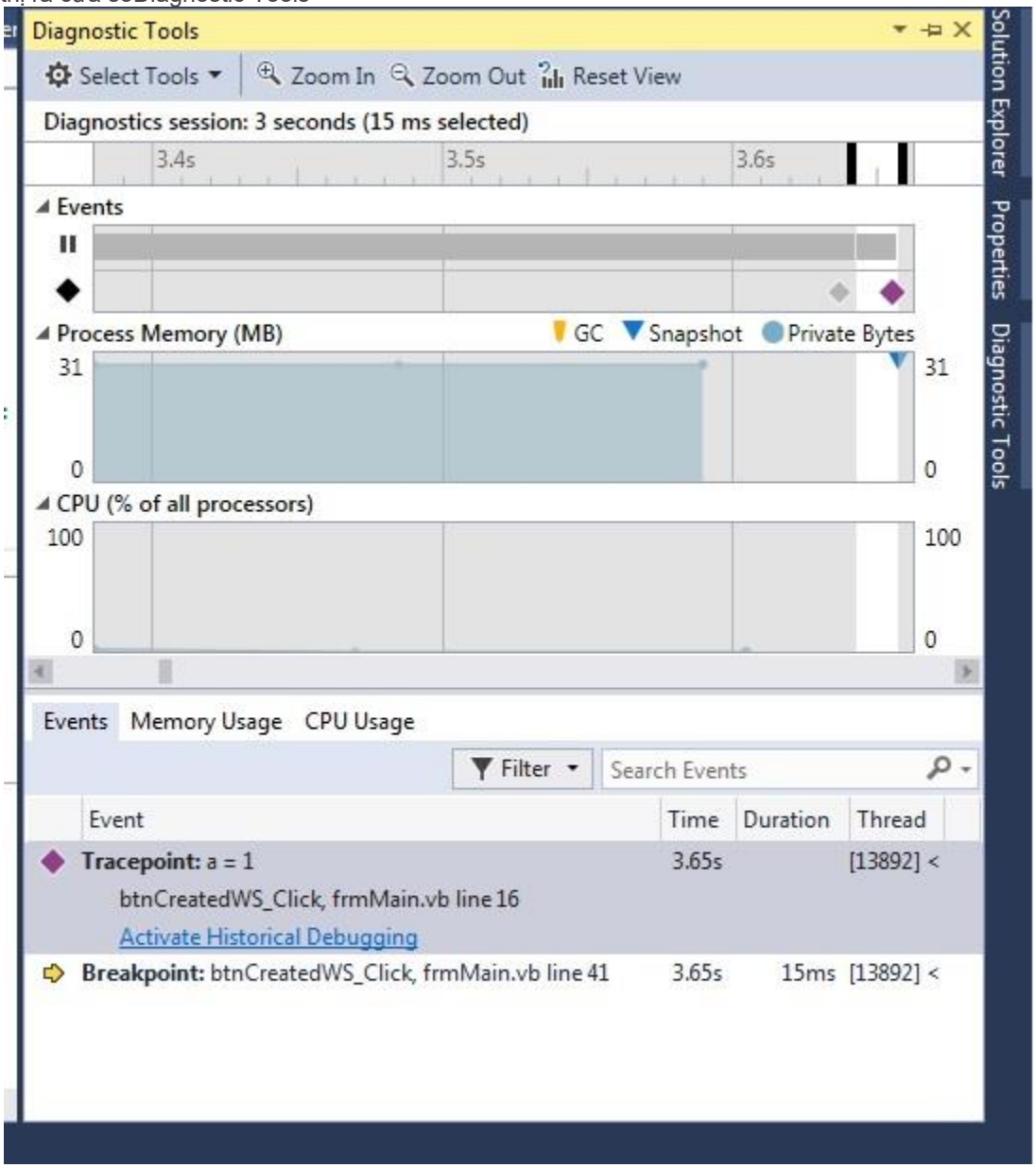
- **PerfTips và Diagnostic Tools**

  Mỗi khi bạn cần tối ưu chương trình, thay vì việc bạn phải đặt timer ở đầu và cuối đoạn code bạn muốn kiểm tra tốc độ thì từ VS 2015 đã được hỗ trợ tính năng là PerfTips (tool**tips** with **perf**ormance information), kèm theo là Diagnostic Tools Window.



Hoặc là bạn có thể dùng **Run to cursor** để biết được cả đoạn code vừa qua tốn bao nhiêu thời gian. Nếu nhấn vào đó VS sẽ hiển

thị ra cửa sổDiagnostic Tools



- **Một vài Tips &Tricks đặc thù**

  Với đặc thù phạm vi các dự án phát triển ở TT PTPM, thông thường các dự án đều có framework và có các tư tưởng thiết kế kiến trúc và việc sử dụng một số thư viện giống nhau. Giả sử đặt địa vị của một lập trình viên vừa mới vào dự án và được nghe giới thiệu về framework của dự án. Tuy nhiên là bạn đó chưa thể nắm ngay được hệ thống kiến trúc đó, mà để nắm được thì phải mất một thời gian ban đầu làm quen dần. Nếu như gặp một vấn đề cần phải debug hoặc là bạn đó muốn tìm hiểu về framework, thì có một số điểm mấu chốt mà các bạn có thể khai thác để tiện hơn cho quá trình Debug của mình.

  - Nếu chức năng có thực hiện cất dữ liệu xuống Database thì các bạn có thể đặt Breakpoint vào hàm khởi tạo Database của đối tượng quản lý việc này. Một số dự án là DLBase, một số dự án có thể là DatabaseFactory, DatabaseUtil,…

- Nếu gặp vấn đề chắc chắn xảy ra Exeption thì bạn có thể đặt Breakpoint vào hàm xử lý Exception của dự án, khi đó mọi Exceptioin của dự án xảy ra bạn đều có thể nhảy luôn vào Debug. Đương nhiên là dự án phải có hàm chung xử lý và yêu cầu tất cả các điểm xảy ra Exception phải gọi vào hàm chung này thì mới xài cách này được
- Tương tự với tình huống có Message cảnh báo lỗi, thì bạn có thể đặt breakpoint vào đối tượng chuyên xử lý Show Message (MISAMessageBox, CommonFunction, …)