> "*Everyone know that **debugging is twice as hard** as w riting a program in the first place.

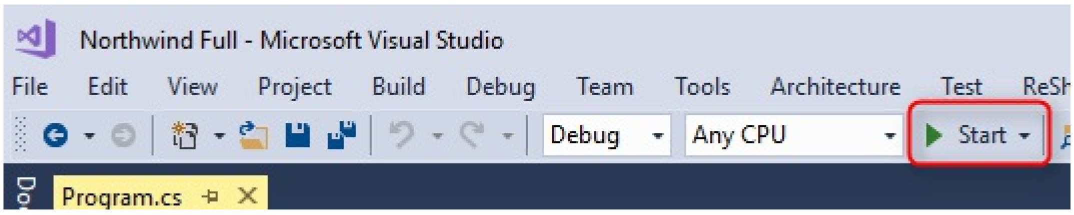
> So if you're as clever as you can be w hen you w rite it, how w ill you ever debug it?*"
>
> - **Brian Kernighan**, Computer Scientist

Introduction

Developers spends more time maintaining and debugging the applications than w riting it.
Visual Studio provides a lot of pow erful tools to debug our application. Some of tools are hidden and are less familiar by many .NET developers.
This article reveals some of the hidden features of Visual Studio and introduces advanced techniques to help you master Visual Studio debugging.

Start Debugging

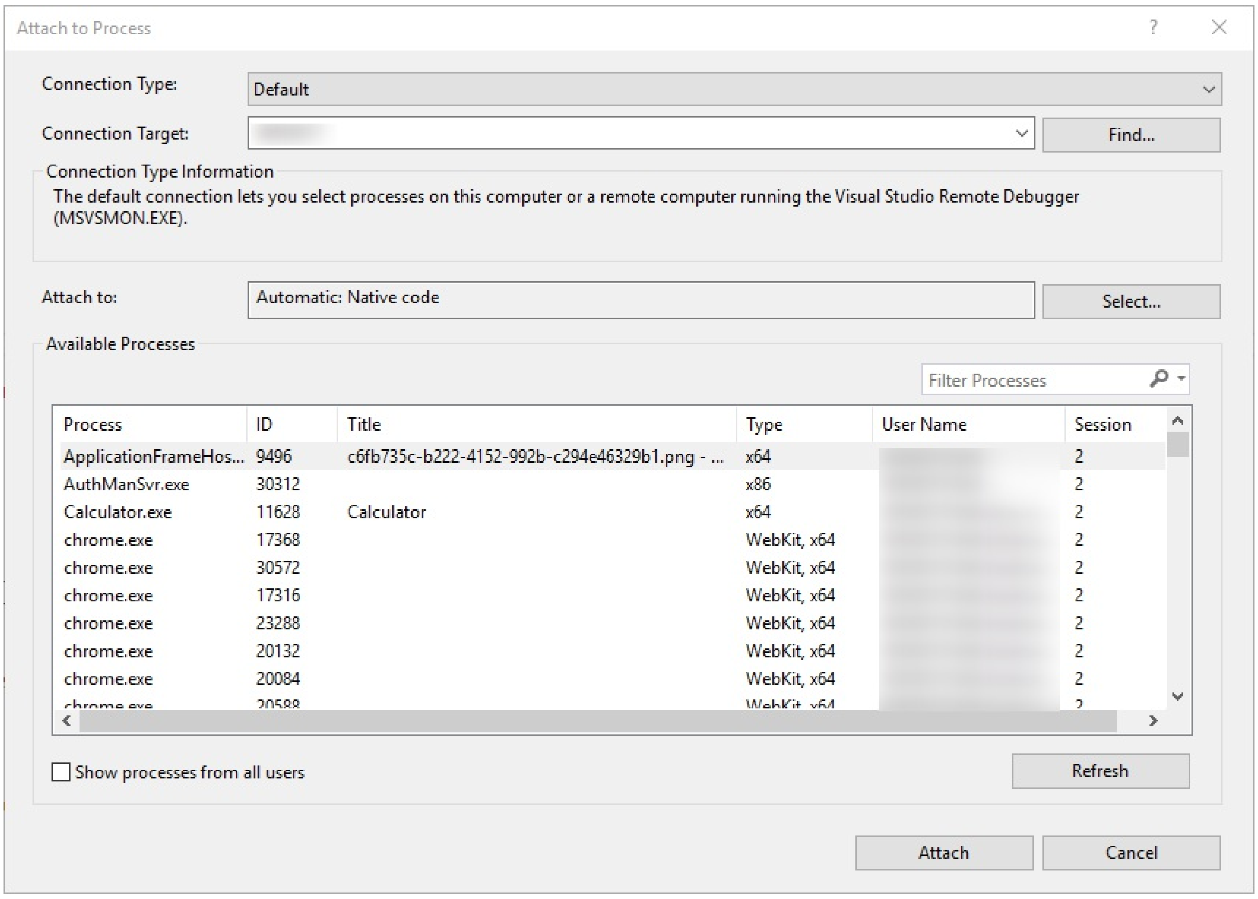
Usually, you start a debugging session by pressing F5 (Debug > Start Debugging), or by clicking the green arrow to start the application w ith the debugger attached.



Once the debugger is attached, you can use the application to navigate to any program and press `Shift + F12` to suspend execution and go to the current running program. This option w orks only in applications that w ere migrated by Firelfy.

Other w ays to start the application w ith the debugger attached are F10, F11, or by right clicking a code line and using Run to Cursor (or pressing `Ctrl + F10`).

If you start the application w ithout debugging you can still attach the debugger to a running process by pressing `Ctrl+Alt+P` (Debug > Attach to Process…) and select the process of the applicaiton from the list.



Attach to Process

Connection Type: Default

Connection Target: [redacted]

Connection Type Information
The default connection lets you select processes on this computer or a remote computer running the Visual Studio Remote Debugger (MSVSMON.EXE).

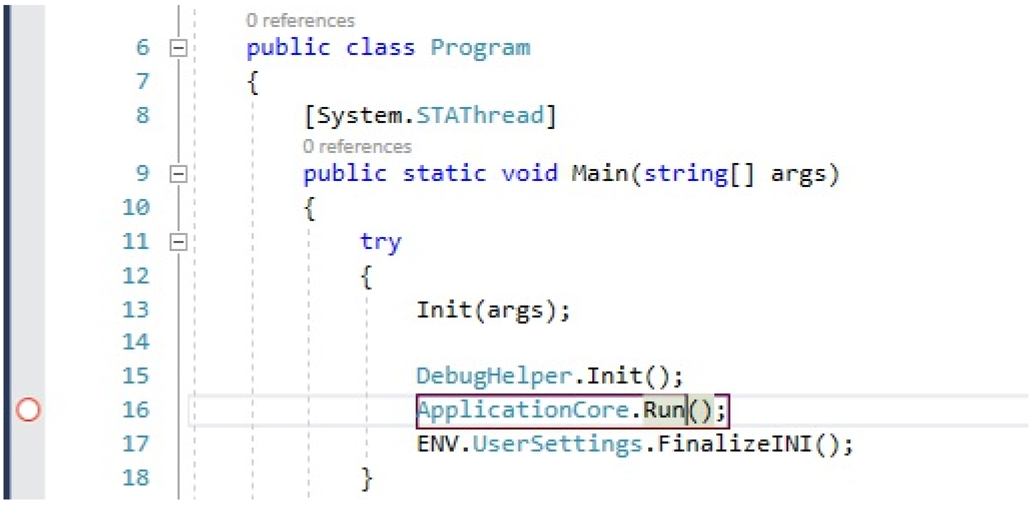
Attach to: Automatic: Native code

Available Processes

Process	ID	Title	Type	User Name	Session
ApplicationFrameHos...	9496	c6fb735c-b222-4152-992b-c294e46329b1.png - ...	x64	[redacted]	2
AuthManSvr.exe	30312		x86	[redacted]	2
Calculator.exe	11628	Calculator	x64	[redacted]	2
chrome.exe	17368		WebKit, x64	[redacted]	2
chrome.exe	30572		WebKit, x64	[redacted]	2
chrome.exe	17316		WebKit, x64	[redacted]	2
chrome.exe	23288		WebKit, x64	[redacted]	2
chrome.exe	20132		WebKit, x64	[redacted]	2
chrome.exe	20084		WebKit, x64	[redacted]	2
chrome.exe	20588		WebKit, x64	[redacted]	2

Breakpoints

You can set breakpoint on any code line by pressing F9, w hen you w ant to suspend the execution.
In case you w ant to disable a breakpoint but keep the location of it, press `Ctrl + F9`.

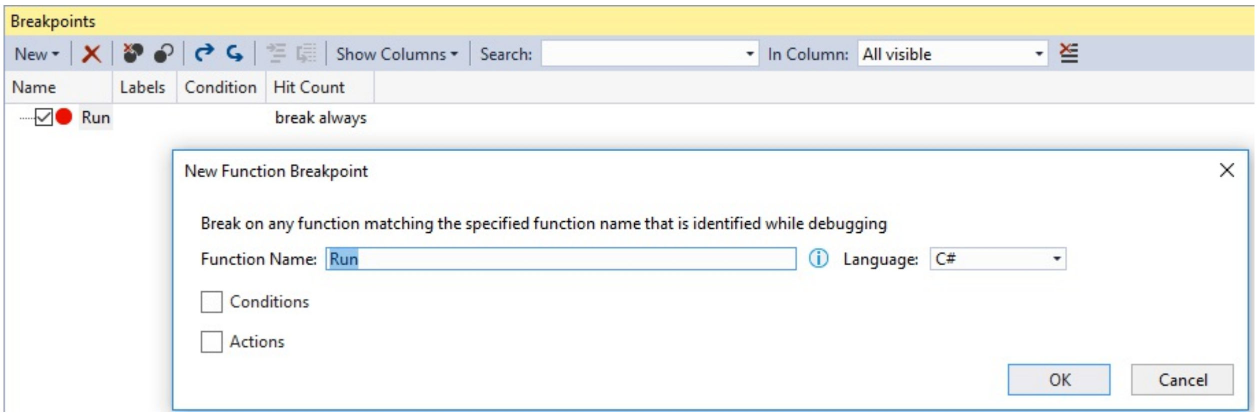


```
0 references
public class Program
{
    [System.STAThread]
    0 references
    public static void Main(string[] args)
    {
        try
        {
            Init(args);

            DebugHelper.Init();
            ApplicationCore.Run();
            ENV.UserSettings.FinalizeINI();
        }
```

Setting a breakpoint from the Breakpoints window

You can also set a new breakpoint from the Breakpoints w indow (Debug > Window s > Breakpoints or `Ctrl + Alt + B`).
This w indow allow s you to manage breakpoints and see all the breakpoints you have set in your application.

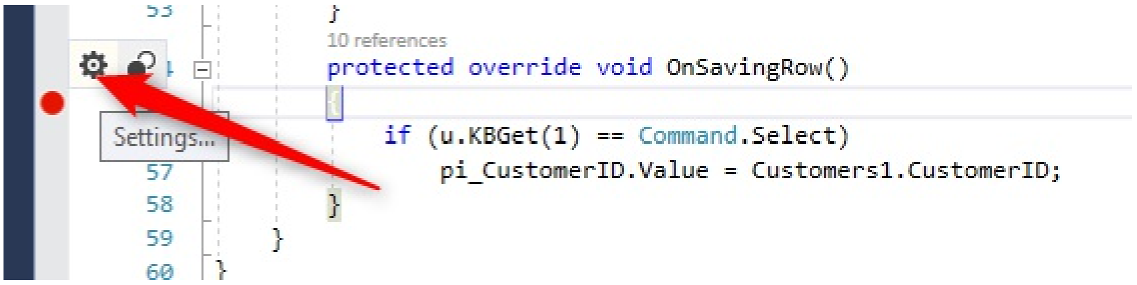
One of the hidden secrets of Visual Studio is the ability to create a new Function Breakpoint from this w indow.
A Function breakpoint is connection to a method name. For example, if you create a new function breakpoint providing "Run" as the function name, the application w ill suspend execution on any call to a method name Run.



This screen gives you options to add conditions and actions to each breakpoint.

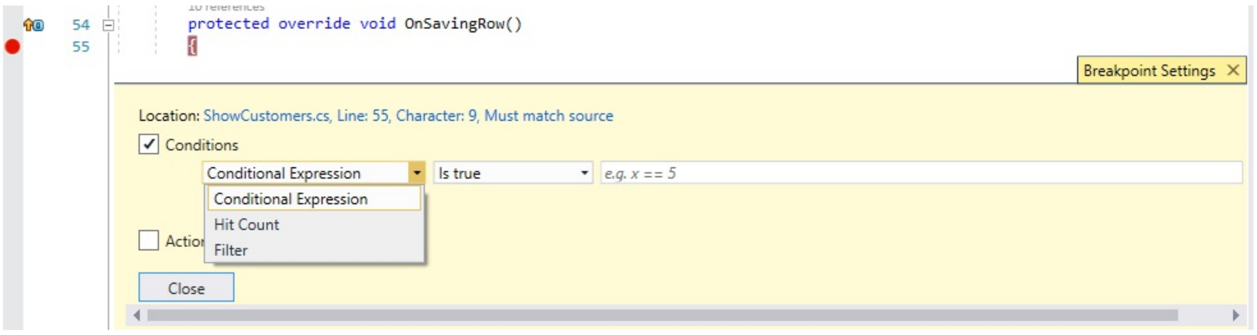
Breakpoint conditions

You can control w hen a breakpoint suspend execution by settings conditions.
Hover over the breakpoint and click on the settings button to open the Breakpoints Settings w indow.



```
    }
    10 references
    protected override void OnSavingRow()
    {
        if (u.KBGet(1) == Command.Select)
            pi_CustomerID.Value = Customers1.CustomerID;
    }
}
}
```

When you check the Conditions box, the w indow expands allow ing you to set conditions for the breakpoint.
You have a few kinds of conditions. Expression that w ill be calculated and break if it is true, or w hen it is changed.
You may also add a hit count condition, w hich is useful to stop inside a loop on a specific iteration.



```
protected override void OnSavingRow()
{
```

Location: ShowCustomers.cs, Line: 55, Character: 9, Must match source

Breakpoint actions

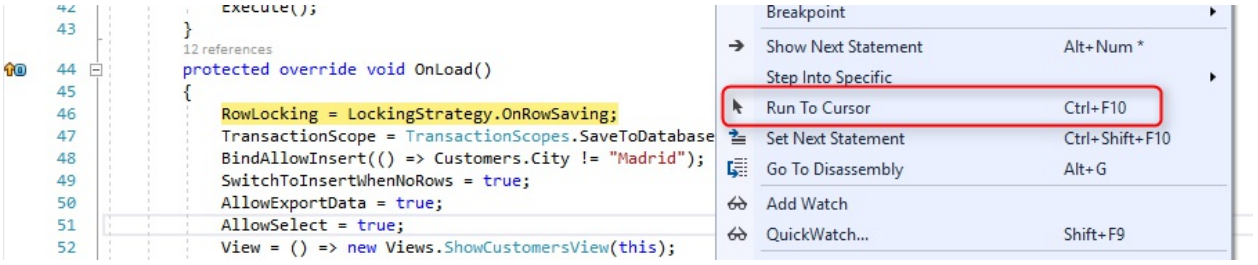
Another hidden feature of Visual Studio is Tracepoints. A tracepoint is a breakpoint that prints a message to the Output w indow.

In the Breakpoint Settings w indow, check the Actions box. You can print any string and also use special keyw ords in the message such as $CALLER, $CALLSTACK, $FUNCTION and more.
See https://docs.microsoft.com/en-us/visualstudio/debugger/using-breakpoints for more information.

Debugging Navigation

Run to Cursor

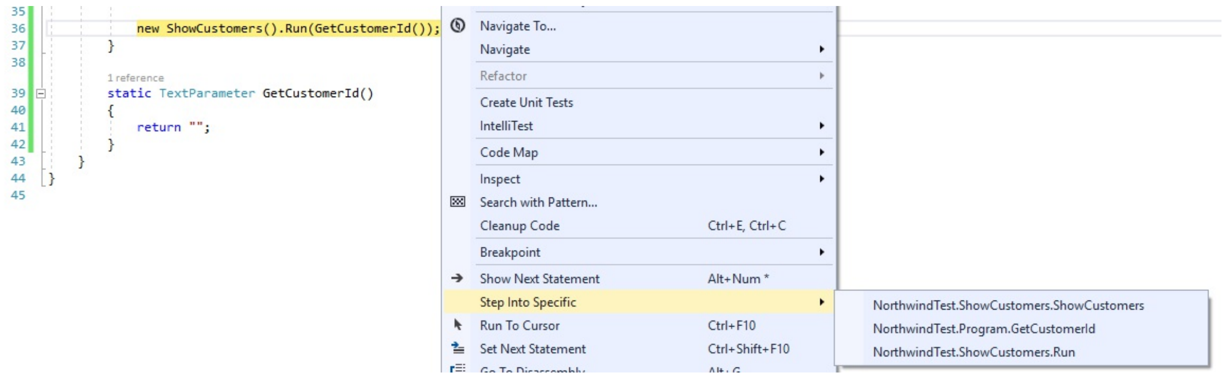
When you debug the application, you can use F10 to step over the code line by line or F11 to step into the code of the method being called.
Sometime you w ant to skip a part of the code and break on a specific line. In this case, set the cursor at the line of code you w ish to stop on, right click and select **Run to Cursor** or press Ctrl+F10



```
            Execute();
        }
        12 references
        protected override void OnLoad()
        {
            RowLocking = LockingStrategy.OnRowSaving;
            TransactionScope = TransactionScopes.SaveToDatabase
            BindAllowInsert(() => Customers.City != "Madrid");
            SwitchToInsertWhenNoRows = true;
            AllowExportData = true;
            AllowSelect = true;
            View = () => new Views.ShowCustomersView(this);
```

Step into Specific

When you have a conplex code line that has more than one method call in it, you can select the specific method you w ant to step into.
Right click the line of code and select **Step into Specific**

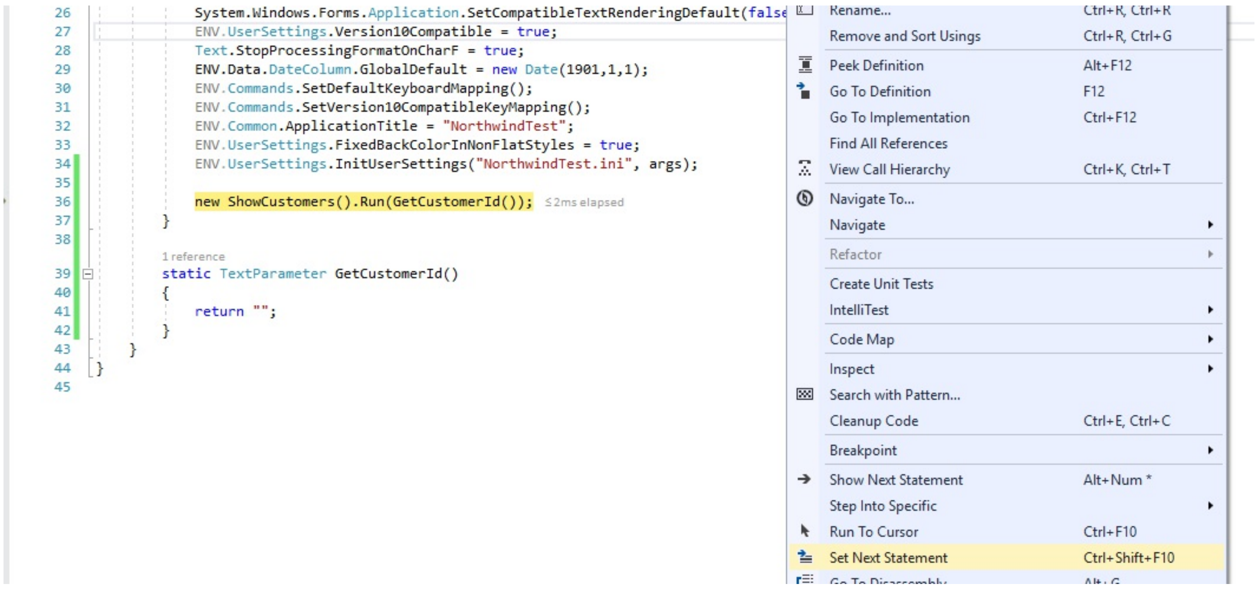


```
            new ShowCustomers().Run(GetCustomerId());
        }

        1 reference
        static TextParameter GetCustomerId()
        {
            return "";
        }
    }
}
```

Navigate To...
Navigate
Refactor
Create Unit Tests
IntelliTest
Code Map
Inspect
Search with Pattern...
Cleanup Code Ctrl+E, Ctrl+C
Breakpoint
Show Next Statement Alt+Num *
Step Into Specific
Run To Cursor Ctrl+F10
Set Next Statement Ctrl+Shift+F10
NorthwindTest.ShowCustomers.ShowCustomers
NorthwindTest.Program.GetCustomerId
NorthwindTest.ShowCustomers.Run

Set next Statement

This option allow you to continue the debugging from any statement you w ant, even going back a few lines and execute them again. Notice that this w ill not undo the last execution but w ill allow you to rexecute the code again.
Right click the line you w ant to execute next and select **Set Next Statement** or press Ctrl+Shift+F10

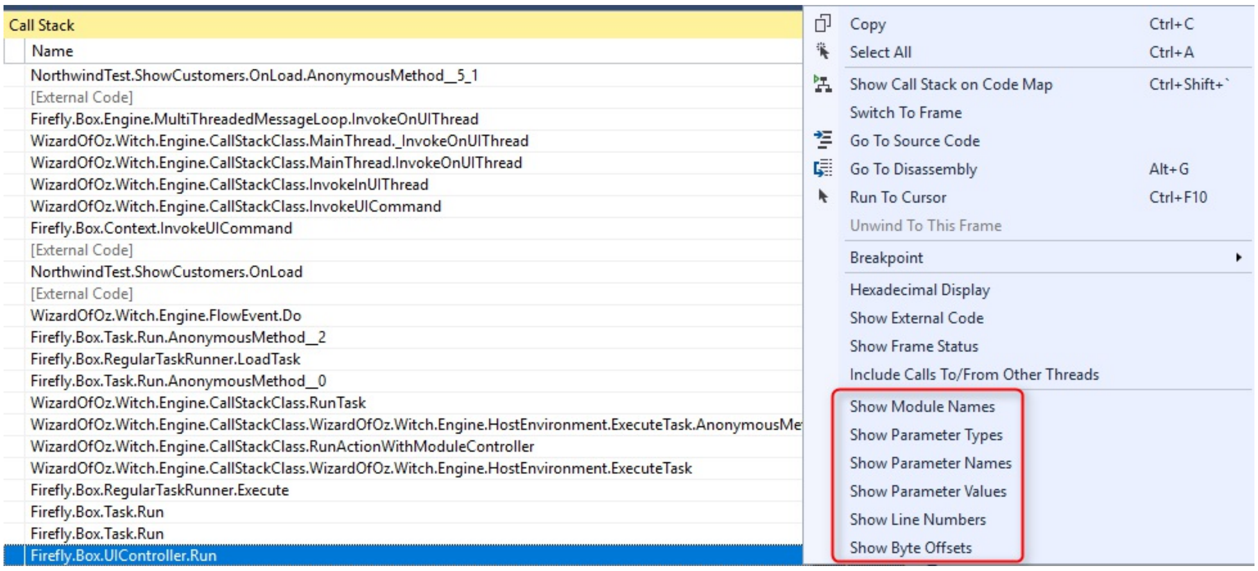


```
            System.Windows.Forms.Application.SetCompatibleTextRenderingDefault(false
            ENV.UserSettings.Version10Compatible = true;
            Text.StopProcessingFormatOnCharF = true;
            ENV.Data.DateColumn.GlobalDefault = new Date(1901,1,1);
            ENV.Commands.SetDefaultKeyboardMapping();
            ENV.Commands.SetVersion10CompatibleKeyMapping();
            ENV.Common.ApplicationTitle = "NorthwindTest";
            ENV.UserSettings.FixedBackColorInNonFlatStyles = true;
            ENV.UserSettings.InitUserSettings("NorthwindTest.ini", args);

            new ShowCustomers().Run(GetCustomerId());  ≤2ms elapsed
        }

        1 reference
        static TextParameter GetCustomerId()
        {
            return "";
        }
    }
}
```

Debug Windows

Call Stack

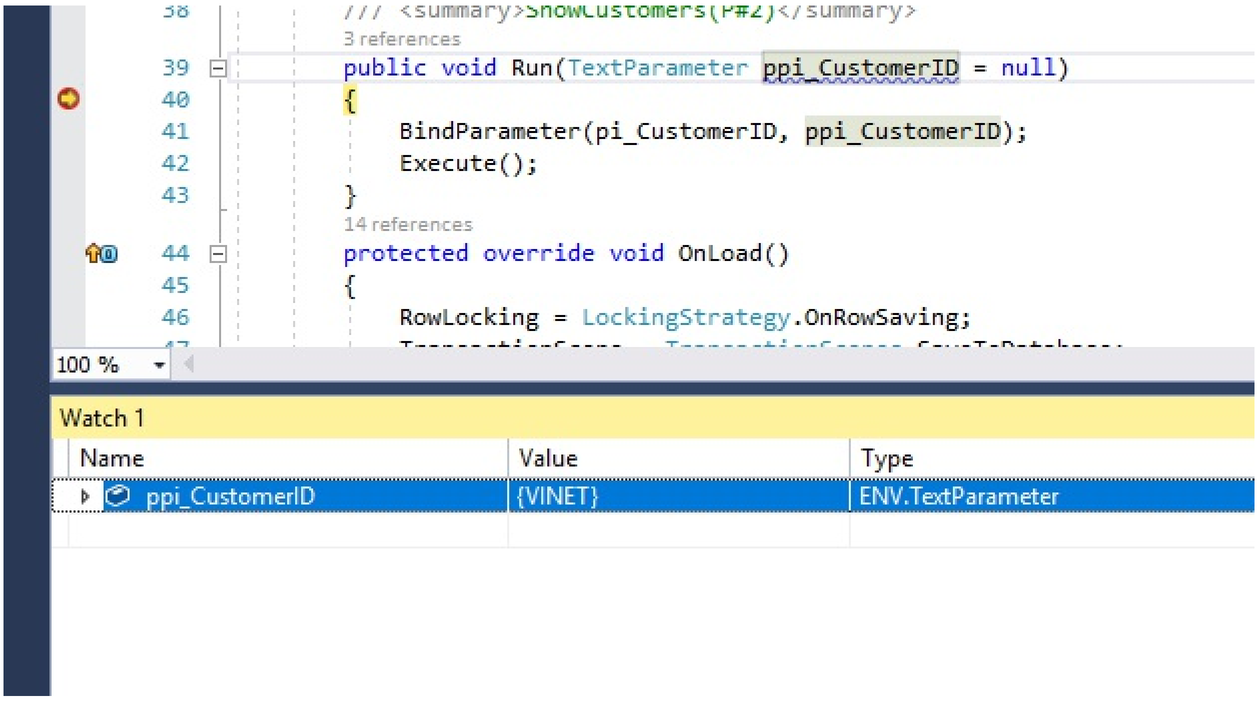
The Call Stack w indow display a list of methods calls that leads to the current program. It is a good w ay to examine and understadnd the execution flow of the application.
Double clicking on any of the lines in the call stack w ill jump to the code of that method.
By default, the call stack displays a lot of information, w hich are usaully not needed, so it is recommended to right click any w here and uncheck all the "Show ..." entries at the bottom of the context menu.



```
Call Stack
Name
NorthwindTest.ShowCustomers.OnLoad.AnonymousMethod__5_1
[External Code]
Firefly.Box.Engine.MultiThreadedMessageLoop.InvokeOnUIThread
WizardOfOz.Witch.Engine.CallStackClass.MainThread._InvokeOnUIThread
WizardOfOz.Witch.Engine.CallStackClass.MainThread.InvokeOnUIThread
WizardOfOz.Witch.Engine.CallStackClass.InvokeInUIThread
WizardOfOz.Witch.Engine.CallStackClass.InvokeUICommand
Firefly.Box.Context.InvokeUICommand
[External Code]
NorthwindTest.ShowCustomers.OnLoad
[External Code]
WizardOfOz.Witch.Engine.FlowEvent.Do
Firefly.Box.Task.Run.AnonymousMethod__2
Firefly.Box.RegularTaskRunner.LoadTask
Firefly.Box.Task.Run.AnonymousMethod__0
WizardOfOz.Witch.Engine.CallStackClass.RunTask
WizardOfOz.Witch.Engine.CallStackClass.WizardOfOz.Witch.Engine.HostEnvironment.ExecuteTask.AnonymousMe
WizardOfOz.Witch.Engine.CallStackClass.RunActionWithModuleController
WizardOfOz.Witch.Engine.CallStackClass.WizardOfOz.Witch.Engine.HostEnvironment.ExecuteTask
Firefly.Box.RegularTaskRunner.Execute
Firefly.Box.Task.Run
Firefly.Box.Task.Run
Firefly.Box.UIController.Run
```

Watch

While debugging, you can use the **Watch** w indow (Debug > Window s > Watch (1, 2, 3, 4)) to w atch variables and expressoins values.
Inside the w atch w indow you can also change the value of any variable and continue debugging using the new value. This is usefull w hen you need to get inside an if statement or pass througe some validaton code.

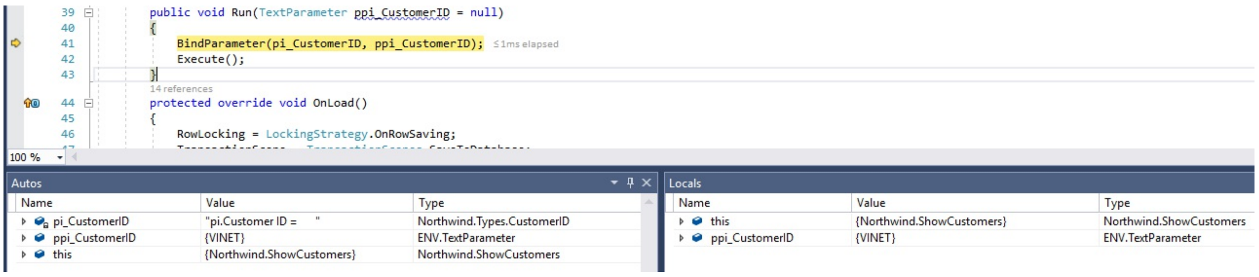


```
/// <summary>ShowCustomers(P#2)</summary>
3 references
public void Run(TextParameter ppi_CustomerID = null)
{
    BindParameter(pi_CustomerID, ppi_CustomerID);
    Execute();
}
14 references
protected override void OnLoad()
{
    RowLocking = LockingStrategy.OnRowSaving;
```

Name	Value	Type
ppi_CustomerID	{VINET}	ENV.TextParameter

Autos and Locals

The **Autos** w indow (Debug > Window s > Autos) and the **Local** w indow (Debig > Window s > Locals) are quite useful w hen you w ant to see variable values w hile you are debugging.
The **Locals** w indow s displays the variables that are defined in the local scope, w hich are the variable that are in use at the current line of code or variable that are part of the current method.
The **Autos** w indow displays variables used around the current line and try to guess w hich variables are more relavent for this specific debug session.
These w indow s somtimes save the need to add variable to the **Watch** w indow manually.



```
public void Run(TextParameter ppi_CustomerID = null)
{
    BindParameter(pi_CustomerID, ppi_CustomerID); ≤1ms elapsed
    Execute();
}
14 references
protected override void OnLoad()
{
    RowLocking = LockingStrategy.OnRowSaving;
```

100 %

Autos

Name	Value	Type
pi_CustomerID	"pi.Customer ID = "	Northwind.Types.CustomerID
ppi_CustomerID	{VINET}	ENV.TextParameter
this	{Northwind.ShowCustomers}	Northwind.ShowCustomers

Locals

Name	Value	Type
this	{Northwind.ShowCustomers}	Northwind.ShowCustomers
ppi_CustomerID	{VINET}	ENV.TextParameter