Debugging Lambda Expressions In Visual Studio 2015

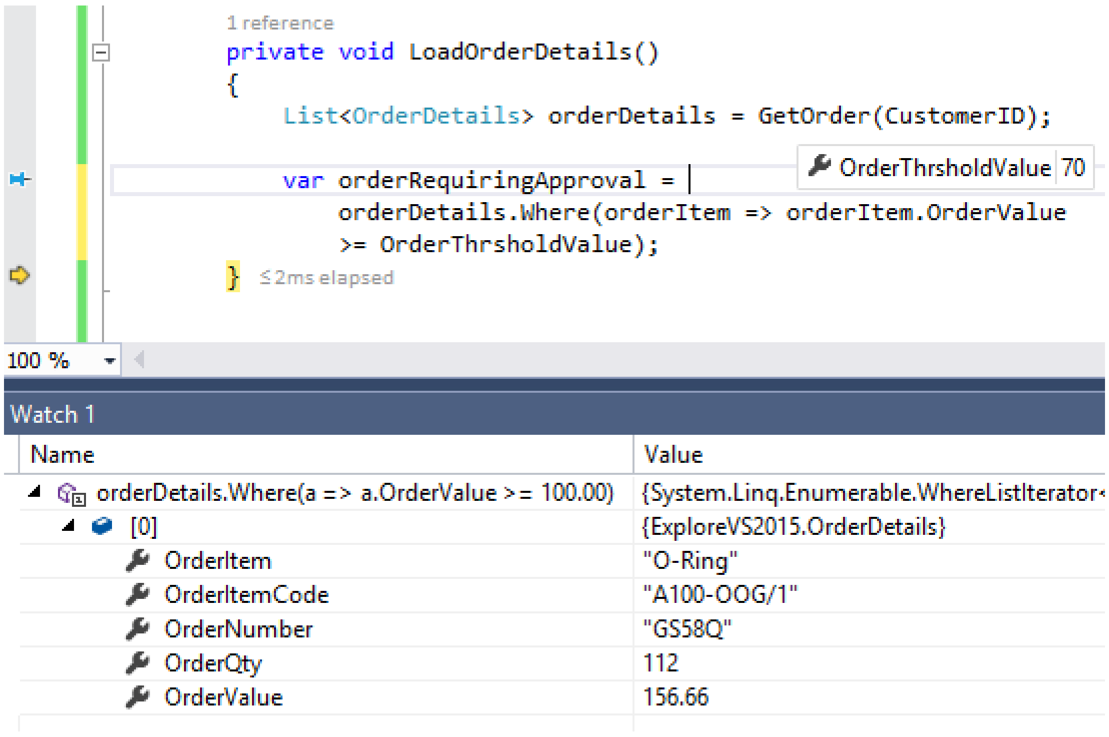
- Programming

Debugging Lambda Expressions – Visual Studio 2015 has added the ability for developers to debug Lambda Expressions. This is a big improvement for me personally because I quite enjoy using Lambda Expressions. Now I can debug these Lambda Expressions and modify values to test different scenarios. Let's take a look how we do it.

Debugging Lambda Expressions

To illustrate how easy it is to debug Lambda Expressions, I have created a simple method that reads order details into a list. As you can see this method returns nothing, but it is just to illustrate the use of Lambda Debugging. I have a Lambda Expression that returns only order items above a certain order value (OrderThreshold of 70). This is useful in a real world application where certain additional requirements are to be set for customers that make orders above a certain value.

From the returned list of orders we can now manipulate the expression in the Watch window. My threshold is 70, but let's see if there are any orders above a higher threshold of 100 for example. We can now type the modified Lambda Expression directly into the Watch window. As you can see, the results are displayed nicely.

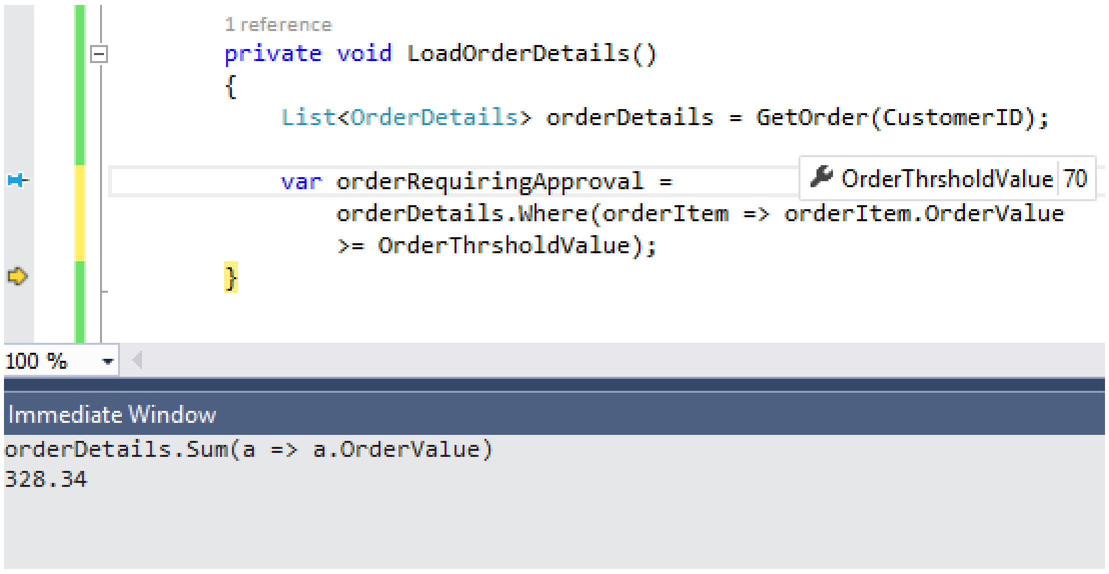


```
1 reference
private void LoadOrderDetails()
{
    List<OrderDetails> orderDetails = GetOrder(CustomerID);

    var orderRequiringApproval =
        orderDetails.Where(orderItem => orderItem.OrderValue
        >= OrderThrsholdValue);
} ≤2ms elapsed
```

```
Name                                          Value
◢ orderDetails.Where(a => a.OrderValue >= 100.00)   {System.Linq.Enumerable.WhereListIterator<
  ◢ [0]                                       {ExploreVS2015.OrderDetails}
      OrderItem                               "O-Ring"
      OrderItemCode                           "A100-OOG/1"
      OrderNumber                             "GS58Q"
      OrderQty                                112
      OrderValue                              156.66
```

Developers can now also do the same in the Immediate Window. Below we are returning the total value of all the orders contained in the order items list.

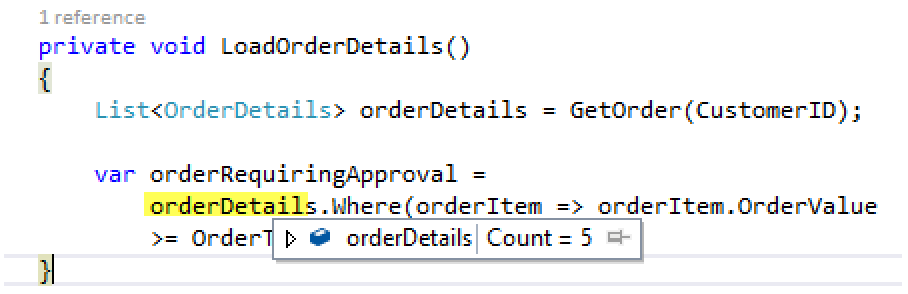


```
1 reference
private void LoadOrderDetails()
{
    List<OrderDetails> orderDetails = GetOrder(CustomerID);

    var orderRequiringApproval =
        orderDetails.Where(orderItem => orderItem.OrderValue
        >= OrderThrsholdValue);
}
```

```
orderDetails.Sum(a => a.OrderValue)
328.34
```

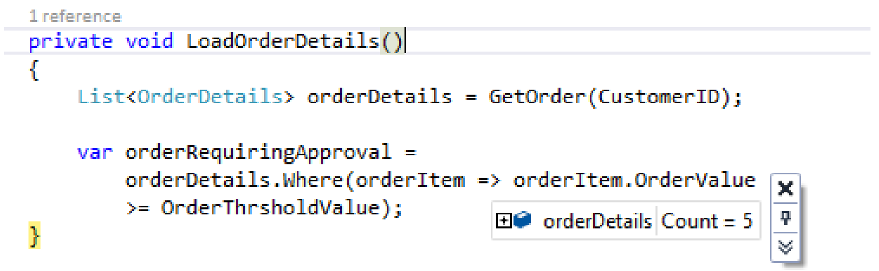
Another nice tidbit is the ability to do the same thing from the pinned data tips. If you hover your cursor over the orderDetails variable, a tip will be displayed that shows you the returned items are 5.



```
1 reference
private void LoadOrderDetails()
{
    List<OrderDetails> orderDetails = GetOrder(CustomerID);

    var orderRequiringApproval =
        orderDetails.Where(orderItem => orderItem.OrderValue
        >= OrderT
}
```

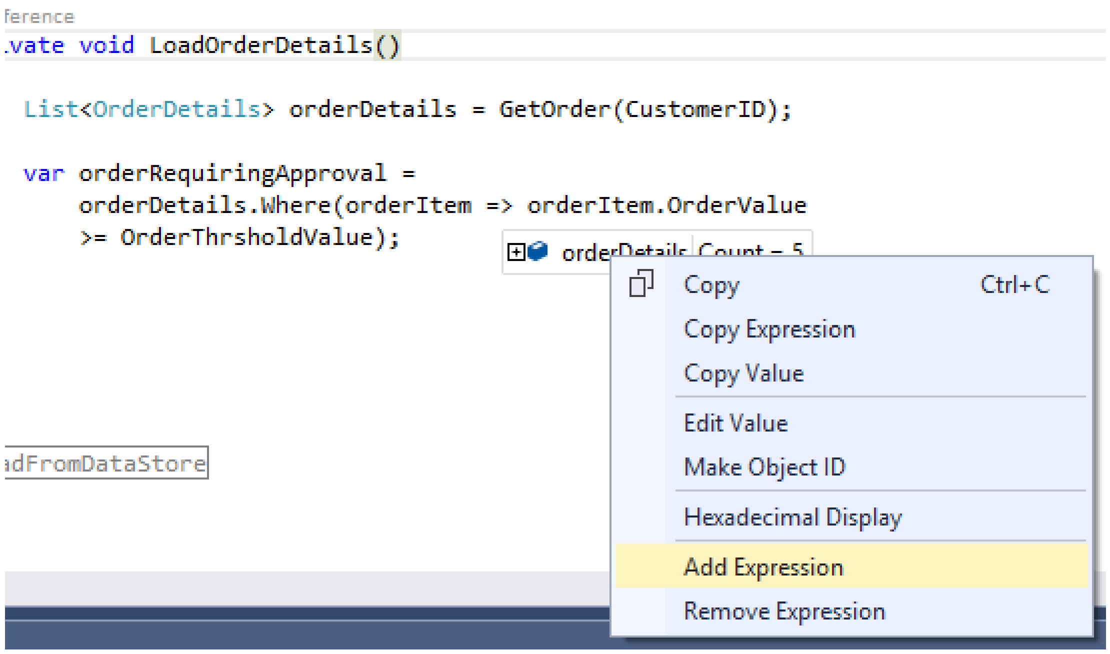
Clicking the push pin will now keep that data tip pinned and always available during your debugging session.



```
1 reference
private void LoadOrderDetails()
{
    List<OrderDetails> orderDetails = GetOrder(CustomerID);

    var orderRequiringApproval =
        orderDetails.Where(orderItem => orderItem.OrderValue
        >= OrderThrsholdValue);
}
```

If you right click this data tip, you can add a Lambda Expression there too.

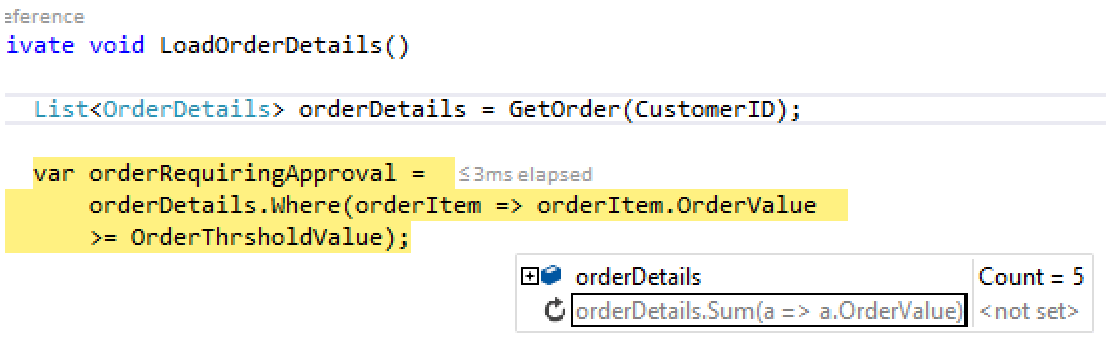


```
ference
.vate void LoadOrderDetails()

 List<OrderDetails> orderDetails = GetOrder(CustomerID);

 var orderRequiringApproval =
     orderDetails.Where(orderItem => orderItem.OrderValue
     >= OrderThrsholdValue);
```

Here I want to get the total value of all the orders (same as we did in the Immediate Window). I can type the Lambda Expression and hit enter.



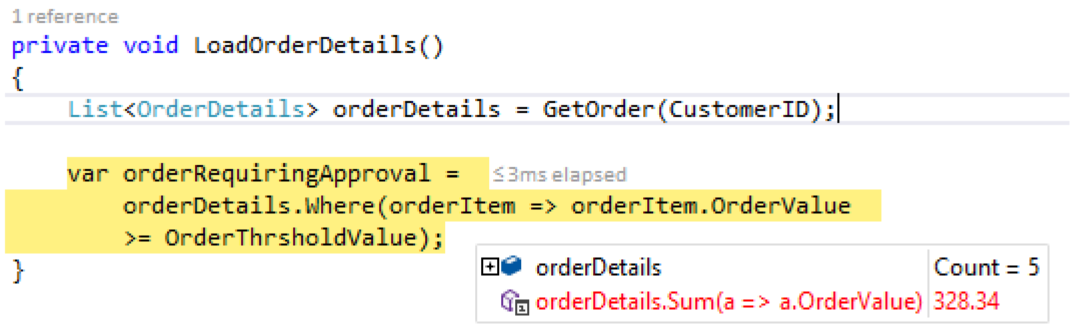
```
eference
ivate void LoadOrderDetails()

  List<OrderDetails> orderDetails = GetOrder(CustomerID);

  var orderRequiringApproval =    ≤3ms elapsed
      orderDetails.Where(orderItem => orderItem.OrderValue
      >= OrderThrsholdValue);
```

orderDetails	Count = 5
orderDetails.Sum(a => a.OrderValue)	<not set>

The Expression is evaluated and the result is displayed in the data tip. This is a convenient solution for any developer several thousand lines deep in code. You are now free to explore, modify and correct faulty Lambda Expressions.



```
1 reference
private void LoadOrderDetails()
{
    List<OrderDetails> orderDetails = GetOrder(CustomerID);

    var orderRequiringApproval =    ≤3ms elapsed
        orderDetails.Where(orderItem => orderItem.OrderValue
        >= OrderThrsholdValue);
}
```

orderDetails	Count = 5
orderDetails.Sum(a => a.OrderValue)	328.34

While the example of the above order items might not have contained big values or contained many orders, consider the usefulness of this Lambda Debugging feature on results containing a couple of hundred records with order values ranging from a couple of thousand to a few million. The ability to debug Lambda Expressions cuts through all this data and allows developers to fine tune the expression to return the exact result they need.

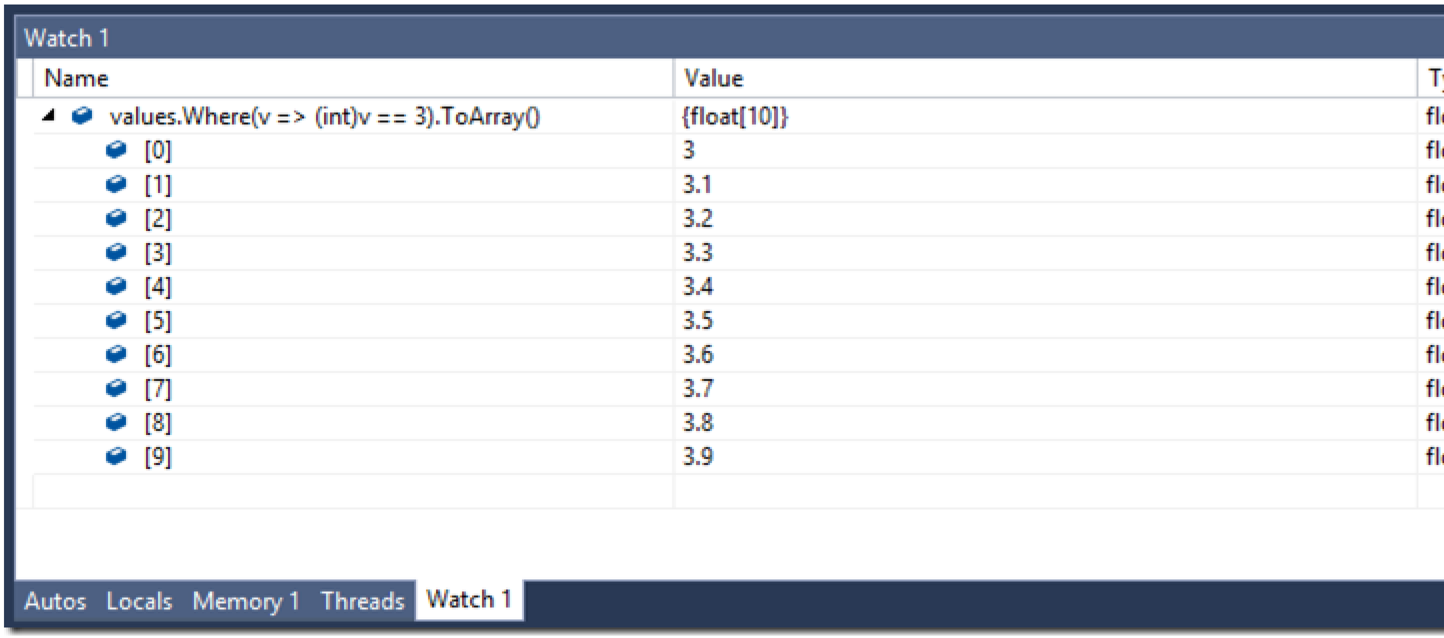
Support for debugging lambda expressions with Visual Studio 2015

[Patrick Nelson - MSFT]

Anyone who uses LINQ (or lambdas in general) and the debugger will quickly discover the dreaded message "Expression cannot contain lambda expressions". Lack of lambda support has been a limitation of the Visual Studio Debugger ever since Lambdas were added to C# and Visual Basic. We've heard your feedback and we are pleased to announce that the debugger now supports evaluation of lambda expressions!

Let's first look at an example, and then I'll walk you through current limitations.

Example



Name	Value	T
values.Where(v => (int)v == 3).ToArray()	{float[10]}	fl
[0]	3	fl
[1]	3.1	fl
[2]	3.2	fl
[3]	3.3	fl
[4]	3.4	fl
[5]	3.5	fl
[6]	3.6	fl
[7]	3.7	fl
[8]	3.8	fl
[9]	3.9	fl

To try this yourself, create a new C# Console app with this code:

```
using System.Diagnostics;
using System.Linq;
```

```
class Program
{
    static void Main()
    {
        float[] values = Enumerable.Range(0, 100).Select(i => (float)i /
10).ToArray();

        Debugger.Break();
    }
}
```

Then compile, start debugging, and add “values.Where(v => (int)v == 3).ToArray()” in the Watch window. You’ll be happy to see the same as what the screenshot above shows you.

NOTE: Lambda expressions that require native functions to run (e.g. LINQ-to-SQL) are not supported.

Happy debugging!