Hands-On Lab

# Debugging with IntelliTrace using Visual Studio Enterprise 2015

Lab version: 14.0.25123.0

Last updated: 4/28/2016



1. **TABLE OF CONTENT**

[Debugging with IntelliTrace using Visual Studio Enterprise 2015 1](#_Toc430703143)

[Overview 3](#_Toc430703144)

[Prerequisites 3](#_Toc430703145)

[Exercises 3](#_Toc430703146)

[Exercise 1: Introduction to IntelliTrace 4](#_Toc430703147)

[Task 1: Configuring IntelliTrace Events 4](#_Toc430703148)

[Task 2: Using IntelliTrace Events to Debug 7](#_Toc430703149)

[Exercise 2: Using IntelliTrace with Call Information 14](#_Toc430703150)

[Task 1: Configuring IntelliTrace to Record Call Information 15](#_Toc430703151)

[Task 2: Using Call Information to Debug 15](#_Toc430703152)

## Overview

* 1. In this lab, you will learn about the IntelliTrace debugging feature in Visual Studio Enterprise. Before the addition of IntelliTrace, debugging was done from a specific point in time going forward. If you wanted to find out more information for a previous call, you would set a breakpoint and then re-run the application. IntelliTrace enables you to seek to a large number of execution points, that occurred in the past, and drill into any of them, comparing variables, call stacks, etc. in order to help pinpoint causes of issues without having to reproduce the problem.
  2. **Note:** This virtual machine has been pre-configured to archive source and symbols for the builds you are working with. This ensures that the files you get back from an IntelliTrace session can be correctly matched to a particular build. In order to best take advantage of IntelliTrace for your software applications, you should review [this blog post](http://www.edsquared.com/2011/02/12/Source+Server+And+Symbol+Server+Support+In+TFS+2010.aspx) which goes into more detail and provides instructions for enabling this. You can also refer to [this](https://msdn.microsoft.com/en-us/library/ms241613.aspx) MSDN article on how to specify symbol locations in the Visual Studio Debugger.

### Prerequisites

In order to complete this lab you will need the Visual Studio 2015 virtual machine provided by Microsoft. For more information on acquiring and using this virtual machine, please see [this blog post](http://aka.ms/almvm).

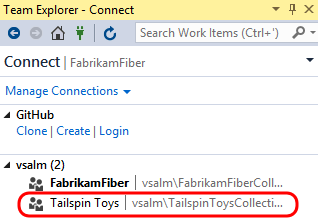
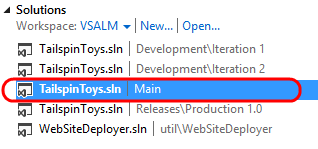
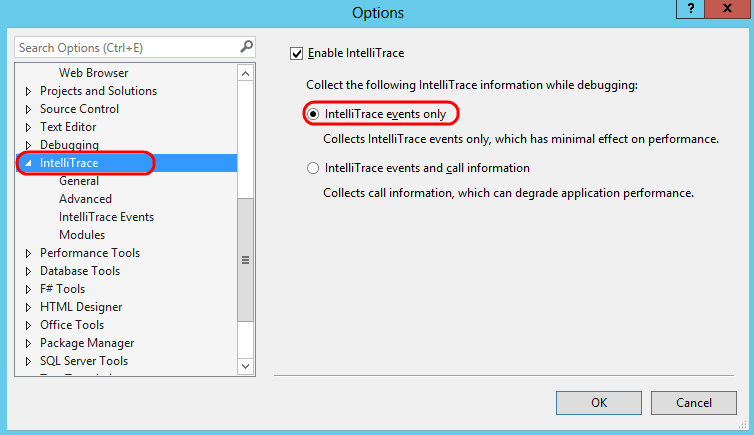
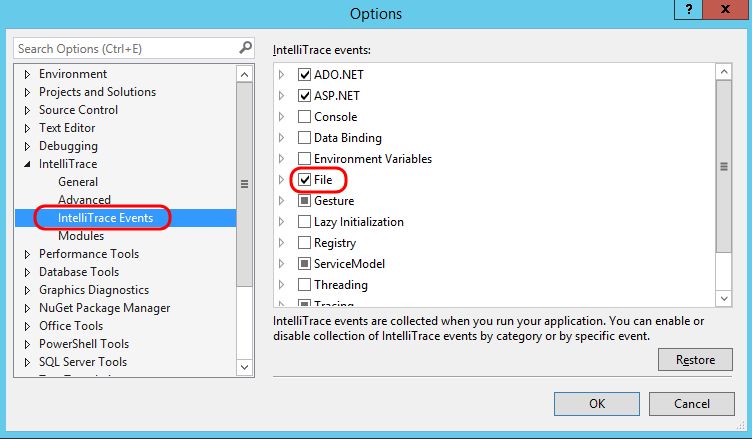
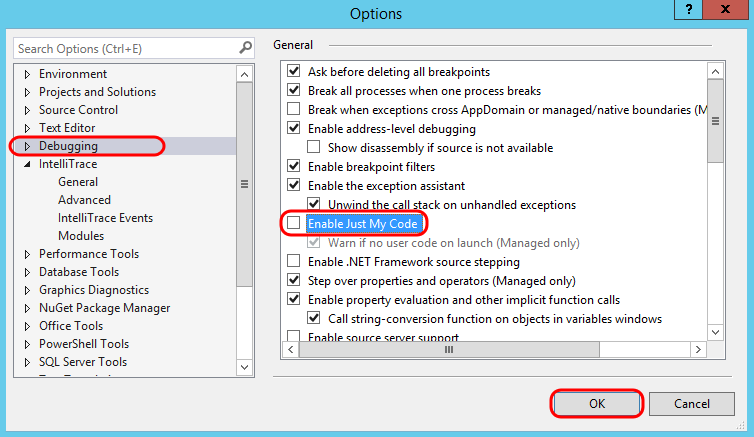
### Exercises

* 1. This hands-on lab includes the following exercises:
  2. Introduction to IntelliTrace
  3. Using IntelliTrace with Call Information
  4. Estimated time to complete this lab: **30 minutes**.

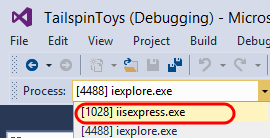
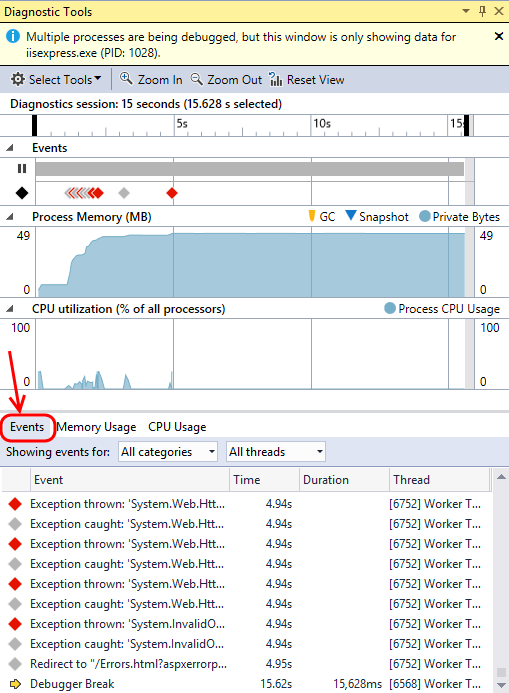
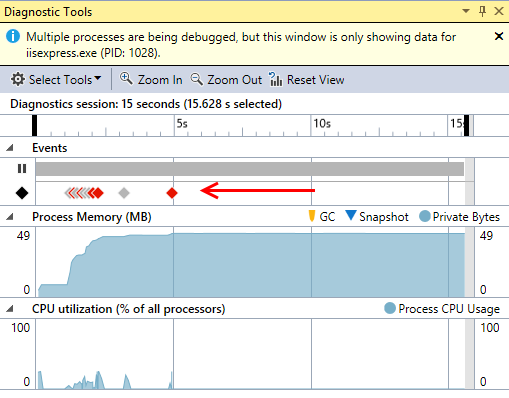
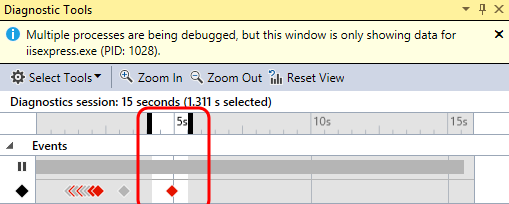
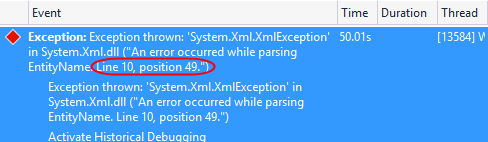
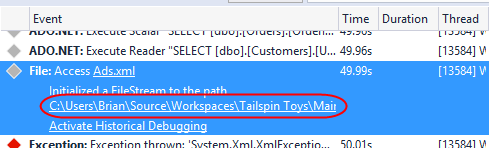
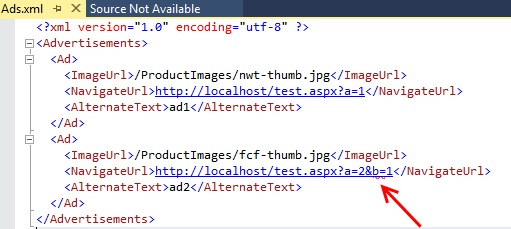
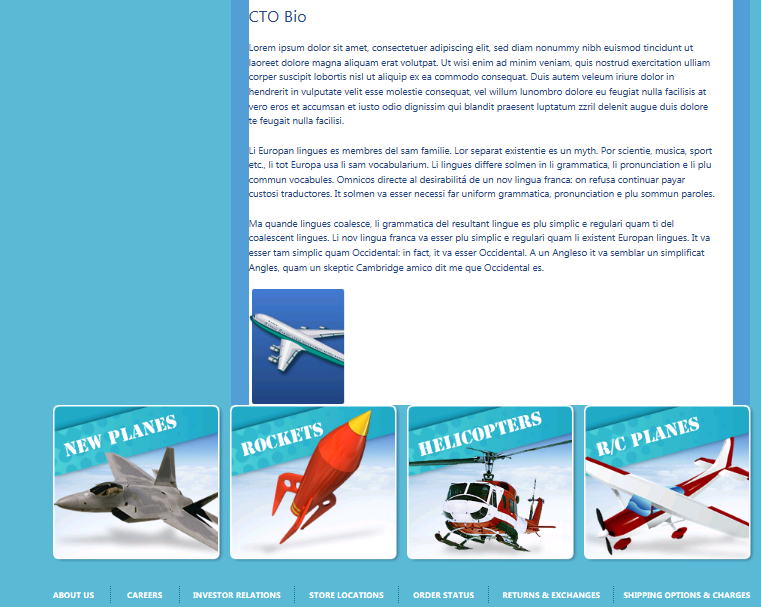
## Exercise 1: Introduction to IntelliTrace

* 1. In this exercise, you will see how IntelliTrace can help debug an XML data failure on an ASP.NET website.
  2. **Note:** IntelliTrace supports debugging Visual Basic and C# applications that use .NET version 2.0 or higher (ASP.NET, Windows Forms, WPF, Windows Workflow, SharePoint 2010, and WCF). IntelliTrace does not support debugging C++, script, or non-managed languages. For more detailed information about IntelliTrace support, please refer to the Visual Studio documentation on the [Microsoft Developer Network](https://msdn.microsoft.com/en-us/library/dd264915.aspx).

#### Task 1: Configuring IntelliTrace Events

* 1. Log in as **Brian** (VSALM\Brian). All user passwords are **P2ssw0rd**.
  2. Launch **Visual Studio 2015** from the taskbar and open **Team Explorer**.
  3. Select the **Connect to Team Projects** button.
     1. 
     2. Figure 1
     3. Connecting to a different team project
  4. In **Team Explorer – Connect**, double-click on the **Tailspin Toys** project.
     1. 
     2. Figure 2
     3. Loading the Tailspin Toys project
  5. In Team Explorer – Home, **double-click** on the third **TailspinToys** solution (associated with the **Main** branch).
     1. 
     2. Figure 3
     3. Loading Tailspin Toys solution
  6. Rebuild the solution (**Build | Rebuild Solution** from the main menu). This step will take a few moments to complete.
  7. Open the **Options** window from **Tools | Options...** and click on the **IntelliTrace** node.
  8. Ensure that the **Enable IntelliTrace** checkbox is checked and that it is set to “**IntelliTrace events only**”. This is the default setting because it has minimal performance implications.
     1. 
     2. Figure 4
     3. IntelliTrace options
     4. **Note:** IntelliTrace records events to an IntelliTrace file, which has an .iTrace extension. These files can be saved and used later in order to perform debugging with IntelliTrace, enabling testers to attach this detailed debugging information directly to bug reports.
  9. Select the **IntelliTrace Events** node on the left and then select the **File** check box to have IntelliTrace record file events.
     1. 
     2. Figure 5
     3. Collect the IntelliTrace events related to file access
  10. Select the **Debugging** node on the left and then select **de-select** the **Enable Just My Code option**. We would like to record events that occur in the .NET Framework code as well. Select the **OK** button.
      1. 
      2. Figure 6
      3. Disabling “Just My Code” feature

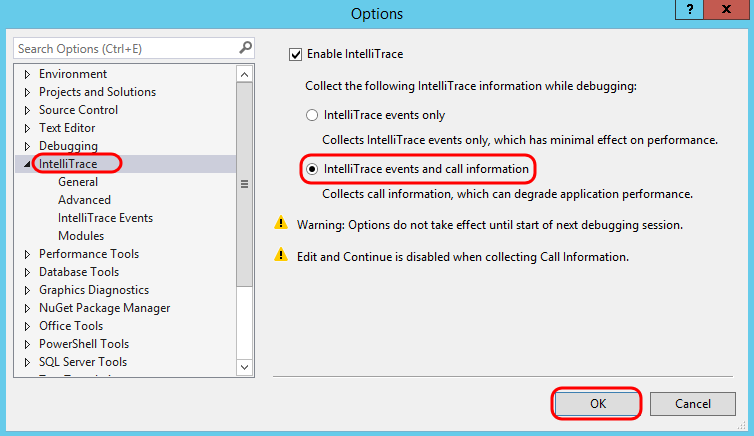
#### Task 2: Using IntelliTrace Events to Debug

* 1. Start a debugging session by pressing **F5**. This will load the default page for the fictional Tailspin Toys company.
     1. **Note:** If the Tailspin.Web project is not set as the startup project, right-click on the Tailspin.Web node in Solution Explorer and select the “Set as StartUp Project” option.
     2. 
     3. Figure 7
     4. Tailspin Toys Default page
  2. Click **About Us** in the footer of the page. This will generate an exception and show a custom ASP.NET error page for the Tailspin Toys application.
     1. 
     2. Figure 8
     3. Location of About Us hyperlink
     4. 
     5. Figure 9
     6. Error Page shown when loading About Us
  3. Switch back to Visual Studio, select the **Break All** button.
     1. 
     2. Figure 10
     3. Break All
  4. Since we’re interested in what is going on with the server code and not the script errors on the web page, select the **iisexpress.exe** process from the **Process** drop-down box.
     1. 
     2. Figure 11
     3. Selecting the correct process for debugging
  5. IntelliTrace events used to be shown in their own window, but are now integrated into the **Diagnostic Tools** window within the **Events** tab. As you just saw, the Diagnostic Tools window is shown by default when you debug an application from within Visual Studio.
     1. 
     2. Figure 12
     3. Diagnostic Tools window showing IntelliTrace events
  6. The **Events** window shows a sequential list of events that have been collected since you started the debugging session, with the most recent being at the bottom. Note that these events are also represented on the Diagnostic Tools timeline. Some event categories have different colors, e.g. Exceptions are shown with red color.
     1. 
     2. Figure 13
     3. IntelliTrace window showing recorded events
  7. Note that there are quite a few events shown in the Events tab. Let’s use the timeline to help filter out the older events that aren’t likely to be associated with the error that we saw with the website. Use the mouse to highlight a window of time that just includes the most recent exception.
     1. 
     2. Figure 14
     3. Filtering events from timeline
  8. In the **Events** tab, locate and select event that starts with “**Exception thrown: ‘System.Xml.XmlException**”. You may need to scroll up some in order to locate it. The details of this exception reports that there is a problem with an XML file on line 10, column 49, but it does not tell us which XML file this is in referring to.
     1. 
     2. Figure 15
     3. Exception thrown when parsing an XML file
  9. Locate the **File Access** event that precedes the exception to see the file that was accessed before the exception occurred.
  10. Open the **Ads.xml** file by clicking on the provided link.
      1. 
      2. Figure 16
      3. Loading file from File Access event
  11. Observe that there is a red squiggly line under the “***&b=1***” characters, which are invalid in this context since the “&” should be escaped as “&amp;”. Note that this is on line 10, column 49 just as the XML exception reported.
      1. 
      2. Figure 17
      3. Ads.xml showing invalid characters
  12. Remove the “&b=1” altogether and **save** the file. The final URL should be **http://localhost/test.aspx?a=2**.
  13. Press **Ctrl+Shift+F5**  (or use Debug | Restart from main menu) to restart the debugging session and select the **About Us** link again to make sure it is now working as expected.
      1. 
      2. Figure 18
      3. About page with working ad
  14. Stop the current debugging session in Visual Studio (**Debug | Stop Debugging**).
  15. Disable the **File** IntelliTrace events that you previously enabled by returning to **Tools | Options | IntelliTrace | IntelliTrace Events**. We will not need these extra events in the next exercise.
  16. **Re-enable** the “**Just My Code**” option before moving on to the next exercise.

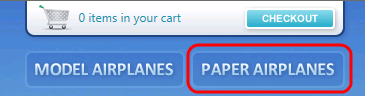
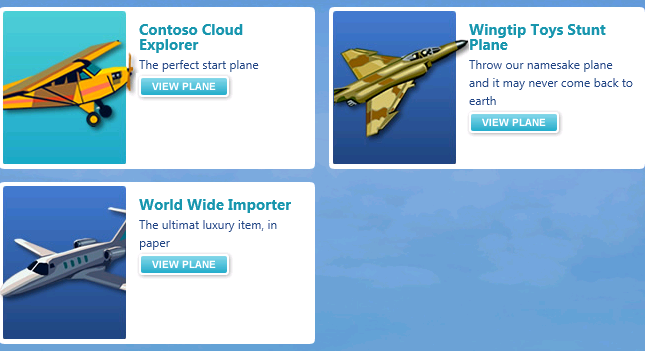
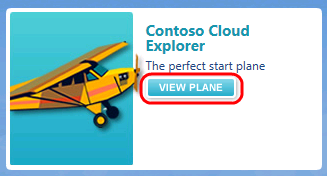
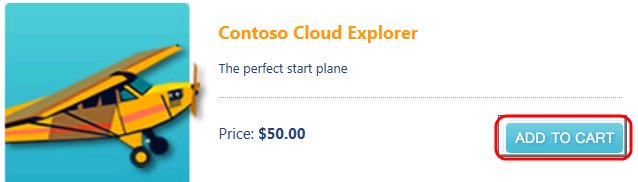
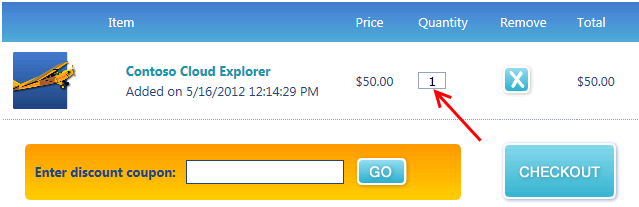
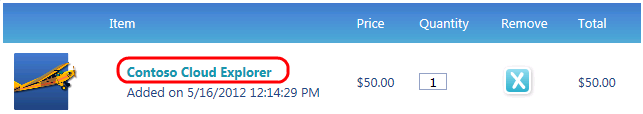
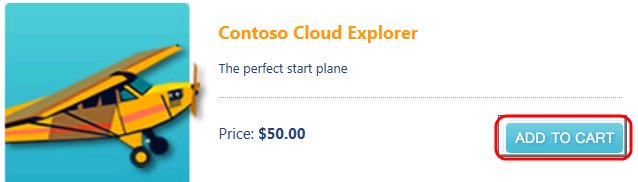
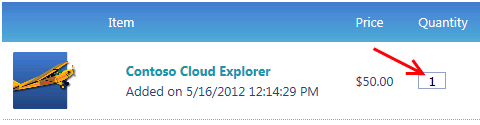
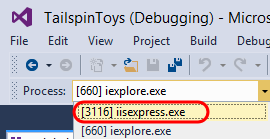
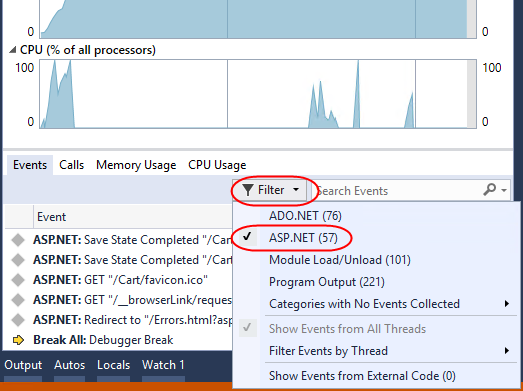
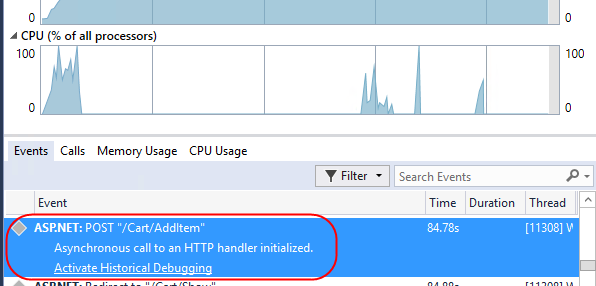
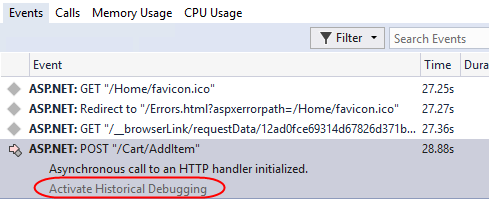
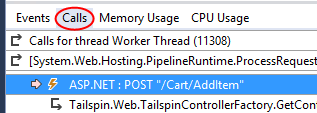
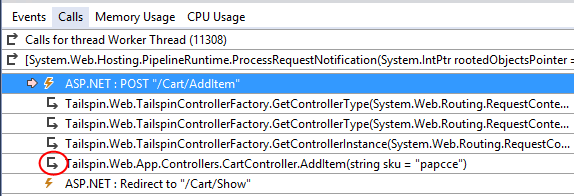
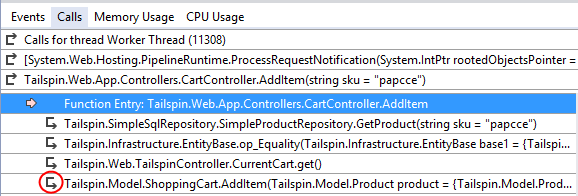
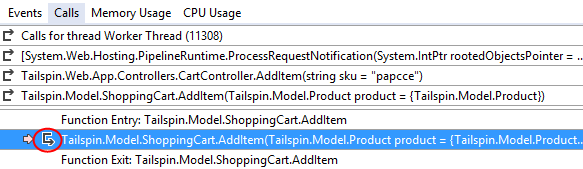
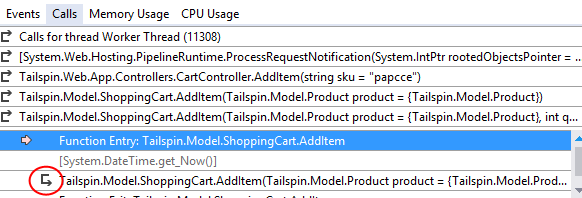
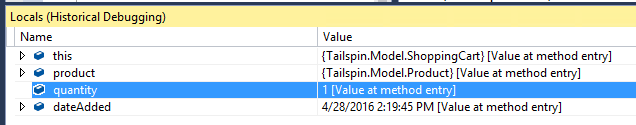
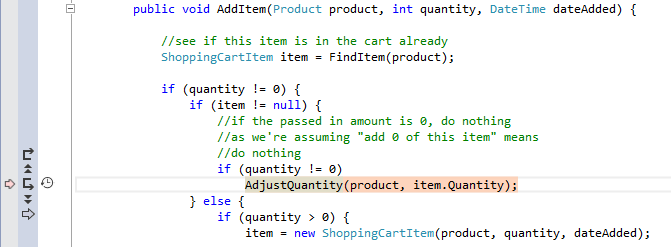
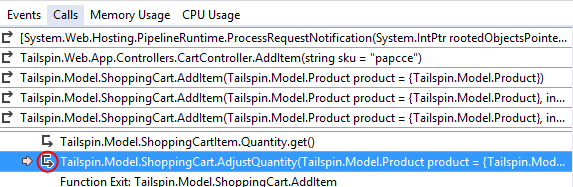
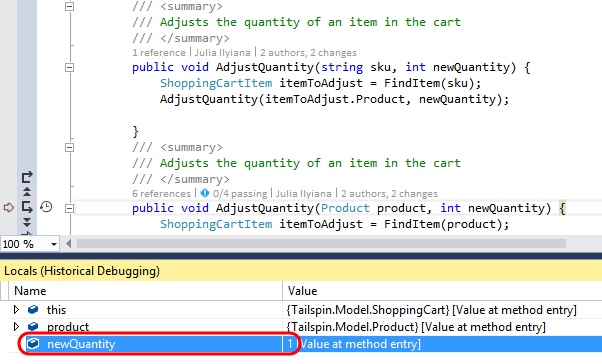
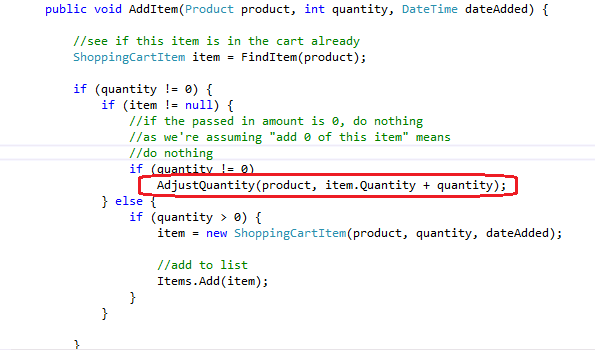
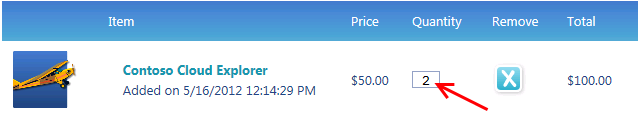
## Exercise 2: Using IntelliTrace with Call Information

* 1. In this exercise, you will learn how to use IntelliTrace to collect additional call information in order to solve a more complicated user code error.
  2. **Note:** Ensure that you have re-enabled the “Just My Code” feature as instructed at the end of the previous exercise before moving on.

#### Task 1: Configuring IntelliTrace to Record Call Information

* 1. By default, only IntelliTrace events are collected when debugging, but we can also opt to collect data at every method enter/exit and call site. To enable this, select **Tools | Options** from the main menu in Visual Studio, locate the **IntelliTrace** node, and select the “**IntelliTrace events and call information**” option.
  2. Select the **OK** button to continue.
     1. 
     2. Figure 19
     3. IntelliTrace options adjusted to record call information

#### Task 2: Using Call Information to Debug

* 1. Start a debugging session by pressing **F5**.
  2. Let’s test out the shopping cart functionality by adding a couple of products to the cart. Select the “**Paper Airplanes**” button to see a catalog of planes.
     1. 
     2. Figure 20
     3. Location of Paper Airplanes button
     4. 
     5. Figure 21
     6. Catalog of paper airplanes
  3. Next, select the “**View Plane**” button for the **Contoso Cloud Explorer.**
     1. 
     2. Figure 22
     3. Location of View Plane button
     4. 
     5. Figure 23
     6. Contoso Cloud Explorer product page
  4. Select the “**Add To Cart**” button and verify that the Quantity of the item in the cart is 1.
     1. 
     2. Figure 24
     3. Location of “Add To Cart” button
     4. 
     5. Figure 25
     6. Shopping cart showing 1 airplane
  5. Add another “Contoso Cloud Explorer” to the shopping cart by selecting the “**Contoso Cloud Explorer**” hyperlink.
     1. 
     2. Figure 26
     3. Location of “Contoso Cloud Explorer” hyperlink
  6. Select the “**Add To Cart**” button once again, but note that the **Quantity** value is still 1 even though we added 2 products.
     1. 
     2. Figure 27
     3. Adding second airplane to cart
     4. 
     5. Figure 28
     6. Product quantity in shopping cart is wrong
  7. Return to Visual Studio and select **Break All** button. By default, we see the Events window showing the sequence of recorded events leading up to the debugger break.
  8. Since we’re interested in what is going on with the server code and not the script errors on the web page, select the **iisexpress.exe** process from the **Process** drop-down box.
     1. 
     2. Figure 29
     3. Selecting the correct process for debugging
  9. Use the **Filter** to deselect all but the **ASP.NET** category.
     1. 
     2. Figure 30
     3. Filtering on ASP.NET events
  10. Scroll up from the bottom until you locate the last **POST** event for “**/Cart/AddItem**” to occur. This event is associated with our action of adding the second product to the shopping cart, and we will use this as a reference point to aid in navigating to the associated call stacks.
      1. 
      2. Figure 31
      3. Locating a specific request
  11. Click the link to Activate Historical Debugging.
      1. 
      2. Figure 32
      3. Activating historical debugging
  12. With the **POST** event still selected, click the **Calls** tab.
      1. 
      2. Figure 33
      3. Switching to Calls view
  13. The top frame of the calls view shows the stack for the current request. Locate the **Tailspin.Web.App.Controllers.CartController.AddItem(string sku = “papcce”)** and click the **Events occurred below this point** button.
      1. 
      2. Figure 34
      3. Location of AddItem call
      4. **Note:** Each time you double-click on a call in the bottom half of the calls view, the call is popped to the bottom of the top frame and the Instruction Pointer syncs in the code editor to the method entry point of the call just like in live debugging.
  14. Locate the “**Tailspin.Model.ShoppingCart.AddItem(Tailspin.Model.Product product = {Tailspin.Model.Product})**” call and click its events button.
      1. 
      2. Figure 35
      3. Navigating call stacks
  15. Locate the call to **AddItem** that shows “**Tailspin.Model.Product product = {Tailspin.Model.Product}, int quantity = 1)**” as the parameter and drill into its events.
      1. 
      2. Figure 36
      3. Navigating call stacks
  16. Finally, locate the call to **AddItem** that shows “**Tailspin.Model.Product product = {Tailspin.Model.Product}, int quantity = 1, System.DateTime dateAdded = …)**” as the parameter and drill into its events. Navigating through the calls view is a quick way to gain an overview of the call hierarchy and to make large jumps across the call hierarchy without stepping through line by line.
      1. 
      2. Figure 37
      3. Navigating call stacks
  17. In the **Locals** window, validate that the correct product was being added to the cart and that the quantity being added was 1.
      1. **Note:** You can look at the collected variables at any point during historical debugging in the Autos/Locals/Watch window or by using the datatips in the editor.
      2. 
      3. Figure 38
      4. Locals window showing parameters during call to AddItem
  18. Step forward two calls by pressing **F10 twice**. Keep in mind that since we are in historical debugging mode, we are stepping through recorded events and call sites, not line by line of code.
      1. 
      2. Figure 39
      3. Location in AddItem method where AdjustQuantity is called
      4. **Note:** You can also use the set of “VCR” style controls shown in the gutter of the source code window.
  19. In the IntelliTrace calls view, drill into the call to **AdjustQuantity**.
      1. 
      2. Figure 40
      3. Navigating call stacks
  20. In the **Locals** window, look to see what quantity the cart is being set to after adding the second product. Since we are expecting the adjusted quantity to be **2**, we have found the location of the bug.
      1. 
      2. Figure 41
      3. Locals window showing incorrect quantity
      4. **Note:** You can also save the IntelliTrace events to a file if needed. Select Debug | IntelliTrace | Save IntelliTrace Session from the main menu. You do not need to save this file for the purposes of this lab, however.
  21. Stop the current debugging session (**Debug | Stop Debugging**).
  22. Change the **AdjustQuantity**(…) call so that it adds in the new quantity passed to the AddItem method.
      1. 
      2. Figure 42
      3. Fixed call to AdjustQuantity method
  23. Verify the fix by adding the same product twice to ensure that the shopping cart quantity shows the value **2**.
      1. 
      2. Figure 43
      3. Shopping cart showing correct quantity of product items

To give feedback please write to [VSKitFdbk@Microsoft.com](mailto:VSKitFdbk@Microsoft.com)

Copyright © 2016 by Microsoft Corporation. All rights reserved.