|  |  |  |
| --- | --- | --- |
| Section 1 | Click Instructions | Talking Points |
| Beacon.pngimage.png  **Click Here** | 1. Open the PartsUnlimited.sln solution in Visual Studio 2017. 2. Click Start to begin debugging. | Business has asked us to determine why we are seeing a drop in the sales of particular items from the past few days. We know that the last deployment was extended functionality to the Search capabilities of our site. Perhaps we can start there and see if this caused an issue?  IntelliTrace is a powerful tool to help locate hard to find bugs. We can choose from two types: F5 or IntelliTrace-in-Production. We'll start with an F5 IntelliTrace session. |
| Beacon.pngimage.png  **Click Here** | 1. Highlight the right hand side of Visual Studio. | As you can see, we're getting all sorts of data in the Diagnostic Tools. |
| Beacon.pngimage.png  **Click Here** | 1. Search for battery. | Our users have complained that some searches aren't returning results. |
| Beacon.pngimage.png  **Click Here** | 1. Search for batteries. |  |
| Beacon.pngimage.png  **Click Here** | 1. In Visual Studio, click the Pause button. | Let's pause execution so we can see what's happening in our application. |
| Beacon.pngimage.png  **Click Here** | 1. CS6: Click Here |  |
| Beacon.pngimage.png  **Click Here** | 1. Open the StyrintContainsProductSearch.cs file. |  |
| Beacon.pngimage.png  **Click Here** | 1. Scroll to the Search method. 2. Right click cleanQuery and select "Search for the Line in IntelliTrace". |  |
| Beacon.pngimage.png  **Click Here** | 1. Highlight the Search results | As you can see, this line was called twice during our exploration. |
| Beacon.pngimage.png  **Click Here** | 1. Click the back arrow to select the first occurence. | Let's go to the first search occurrence. |
| Beacon.pngimage.png  **Click Here** | 1. Point out the VCR controls. | You can use the VCR like controls to step into and over our code execution. |
| Beacon.pngimage.png  **Click Here** | 1. Float over the query parameter. | Note that we can see 'battery' was passed into this method. |
| Beacon.pngimage.png  **Click Here** | 1. Click the step over button a few times, until we get out of the method. | It seems like this returned on a fairly normal path. |
| Beacon.pngimage.png  **Click Here** | 1. Click the right arrow to get to the 2nd execution. | Let's take a look at the next time this line was called. |
| Beacon.pngimage.png  **Click Here** | 1. Step into the code. | This time let's step into the Depluralize method. |
| Beacon.pngimage.png  **Click Here** | 1. We can see the logic that is executing. | It looks like we're replacing the 'ies' in batteries with a single 'y' |
| Beacon.pngimage.png  **Click Here** | 1. Step forward a few more times to leave the method. |  |
| Beacon.pngimage.png  **Click Here** | 1. Click the Locals Historical Debugging tab. |  |
| Beacon.pngimage.png  **Click Here** | 1. Show the local variable values. | By looking at the locals window, we can see the original search term, along with the value returned from the Depluralize method. |
| Beacon.pngimage.png  **Click Here** | 1. Click the stop debugging icon. | Looks like everything is really behaving normally. Instead of us trying to debug this, let's go gather an IntelliTrace log from our production server. |
| Beacon.pngimage.png  **Click Here** | 1. Close Visual Studio. |  |
| Beacon.pngimage.png  **Click Here** | 1. Open Powershell and browse to C:\intellitrace | We have some idea of where the bug may be - in the search functionality. But how do we debug in Production? Ideally, we want some way of stepping through the Production code without affecting production. This means we don't want to attach Visual Studio! However, what if we could create a log that we could open up and "debug" - stepping through the code as if it was live? |
| Beacon.pngimage.png  **Click Here** | 1. Open a PowerShell prompt and enter the following commands: 2. cd \intellitrace 3. Import-Module .\Microsoft.VisualStudio.IntelliTrace.PowerShell.dll | Using IntelliTrace we can capture a log of code running in production. Let's get started! The first thing we have to do is attach an IntelliTrace collector to the running process. We have an IIS server running the Parts Unlimited ASP.NET application on this machine.  This imports the IntelliTrace cmdlets so that we can create a trace of what is happening in production. |
| Beacon.pngimage.png  **Click Here** | 1. Start the collection with the following command. Enter 'y' when prompted to start the trace. 2. Start-IntelliTraceCollection -ApplicationPool DefaultAppPool -CollectionPlan .\collection\_plan.ASP.NET.trace.xml -OutputPath .\logs\ | This starts IntelliTrace, gathering diagnostic logs on the DefaultAppPool. We specified the trace.xml collection plan which will give us a verbose log and have a slightly heavier impact on performance (hence we only want to run this for short periods). For a lighter log, we could have used the default.xml collection plan. |
| Beacon.pngimage.png  **Click Here** | 1. Open IE. |  |
| Beacon.pngimage.png  **Click Here** | 1. Browse to PartsUnlimited. | Now we're going to simulate some activity on the website ourselves. Often, you can just let normal usage drive the IntelliTrace collection, but we're going to have to simulate it here. By the way, attaching the IntelliTrace collector to the app pool recycles it, which is why the first load takes a bit longer. |
| Beacon.pngimage.png  **Click Here** | 1. Enter battery into the search box. |  |
| Beacon.pngimage.png  **Click Here** | 1. Enter batteries into the search box. | Searching for battery and batteries both returned 2 batteries, so that worked. |
| Beacon.pngimage.png  **Click Here** | 1. Enter jumper lead into the search box. |  |
| Beacon.pngimage.png  **Click Here** | 1. Enter jumper leads into the search box. |  |
| Beacon.pngimage.png  **Click Here** | 1. Point out lack of search results. | Interesting. Searching for Jumper Lead returned an item, but searching for Jumper Leads did not. We may have found a problem! |
| Beacon.pngimage.png  **Click Here** | 1. Return to the PowerShell prompt. |  |
| Beacon.pngimage.png  **Click Here** | 1. Enter the following to stop collection: 2. Stop-IntelliTraceCollection -AppPool DefaultAppPool 3. Click Y when prompted. | This stops the trace on the DefaultAppPool so that we don't impact performance any more than necessary. It also removes the lock on the log file. (If we wanted to continue logging we could have used CheckPoint-IntelliTraceCollection to instead release the log file and start logging to a new file). |
| Beacon.pngimage.png  **Click Here** | 1. Open an explorer window to c:\intellitrace\logs |  |
| Beacon.pngimage.png  **Click Here** | 1. Double-click the iTrace file to open it in VS 2017. | Note that there is a new iTrace file in this folder (called something like w3wp\_00001200\_170131\_073315.iTrace). This is the log that the IntelliTrace collector created.  Now that we have an iTrace file, we can open it in Visual Studio and diagnose what is going on in Production. |
| Beacon.pngimage.png  **Click Here** | 1. Show the exceptions, plus the other pieces of data. | We'll see call stacks, locals and variable values as though we were debugging the Production site! |
| Beacon.pngimage.png  **Click Here** | 1. Open the Web Requests folder | We can see all the requests that were logged during the session. |
| Beacon.pngimage.png  **Click Here** | 1. Click the top URL for the 'jumper leads' search. | Let's drill into the detail for this last request. The one that caused the issue. |
| Beacon.pngimage.png  **Click Here** | 1. Click Request Details. |  |
| Beacon.pngimage.png  **Click Here** | 1. Click on the first Exception that is noted in the log. | We can see the exception being thrown - but the site doesn't report an error. This is a really hard scenario to debug if we don't have IntelliTrace since it's an intermittent issue (the search worked for batteries but not for jumper leads). We can see that the exception is a System.ArgumentOutOfRangeException. |
| Beacon.pngimage.png  **Click Here** | 1. Click on the Debug this Event button. |  |
| Beacon.pngimage.png  **Click Here** | 1. Note the highlighted code! | It looks like we're debugging, but in fact this is showing history from the data captured in Production. By clicking on Debug, we are taken to the exact line of code that caused the exception. We can view the Call Stack and locals - and even see what value variables had at this point in the log. |
| Beacon.pngimage.png  **Click Here** | 1. Examine the line more closely. | We now know where the problem is - we're indexing past the length of the string. This is a classic off-by-one error. We can now stop debugging and figure out how to fix the bug. |
| Beacon.pngimage.png  **Click Here** |  | You can spend less time debugging your application when you use IntelliTrace to record and trace your code's execution history. You can find bugs easily because IntelliTrace lets you record specific events and debug errors that are hard to reproduce or that happen in deployment  Here are some examples of how IntelliTrace can help you with debugging:  Your application has corrupted a data file, but you don't know where this event happened: Without IntelliTrace, you have to look through the code to find all possible file accesses, put breakpoints on those accesses, and rerun your application to find where the problem happened. With IntelliTrace, you can see all the collected file-access events and specific details about your application when each event happened.  An exception happens:Without IntelliTrace, you get a message about an exception but you don’t have much information about the events that led to the exception. You can examine the call stack to see the chain of calls that led to the exception, but you can’t see the sequence of events that happened during those calls. With IntelliTrace, you can examine the events that happened before the exception.  Your application crashes on a test computer but runs successfully on a development computer or A bug or crash happens in a deployed application: You can configure IntelliTrace data collection before you publish the application. While your application runs, IntelliTrace saves data to an .iTrace file or use Microsoft Monitoring Agent, either alone or with System Center 2012, to save IntelliTrace data to an .iTrace file. |