**Advanced.net debugging pdf download**

I started a new technical blog that talks about the cloud. It's called Forecast: Cloudy, and it will include thoughts, ideas and some code, driven mostly by my experience of services on cloud infrastructure. While it won't necessarily talk about .NET and debugging it will include general architectural examples as well as use cases to use the various services that are also available from .NET. The first post (after the traditional Hello World) already has code ☺ check it out and be sure to tell me what you think about it. I just got mailed a new Advanced .NET Debugging book by Mario Hewardt which I had the pleasure of reviewing before publishing it (I really hope my comments helped a little, if at all ☺). The book covers internal .NET, tools that can help you fine-tune and solve such issues, how, WinDbg and PowerDBG, also cover the specific and common issues that many other developers have encountered (heaps of corruption, dead locks, etc.) the book is well written and structured, and I would recommend it to any .NET developer who would like to better understand how .NET works under the hood, as well as get the tools they need (and it's necessary to use these tools ☺) to fix and solve the real world problems that can't be solved with regular debugging, such as Visual Studio. The book fills a niche that has remained empty for too long. When I started this blog there was almost no knowledge on the internet about advanced .NET debugging. I'm glad that Mario was the one to bring all this information into a structured book. Get the book here: You can also check out our previous book Mario, Advanced Windows Debugging, which also talked about using WinDbg and other tools, but in a native context. It's also a great book in addition to the .NET side of things. Get the book and debug without fear! ☺ If you're using AJAX management tools, you've probably noticed ToolkitScriptManager's server management. One of the features of ToolkitScriptManager that comes with AJAX Management Tool is its ability to combine all the client's side scenarios, which must be downloaded to the browser in one query, thereby saving the browser the need to issue multiple requests to the server and speed up the page loading process. The URL of this combined query contains the hash code of each script that needs to be merged, so if you have different combinations of client side scripts because of the different combinations of controls that you use from ajax Controlkit Tool, you will have a unique URL for each request. So what's in it for me?, you ask. It's simple. Because these scripts only change when you change the AJAX Control Toolkit version (and it only happens from time to time), and these scripts can be quite large (even when Gziped), they are ideal candidates for delivery from the network (CDN). In short, CDN can make your site faster by providing static content (i.e. images, scripts, static html files, etc.) etc.) etc.) Place closer to the user by making a request. It does this by making a request on behalf of the user, requesting that resource, caching it over a period of time, and distributing it to a server that is geographically closer to the user. In most CDN systems, you need to change the host name of the resource you want from CDN to something that was pre-configured to work with your site. The problem with the AJAX management tool is that it displays the link to the combined javascripts, and you can't control it. Fortunately, The Designers of the ToolkitScriptManager class thought about a hook that allows you to change the URL of the combined script handler. If you install a CombineScriptsHandlerUrl property with a URL that refers to CDN, you'll make those scripts downloaded via CDN, thus making your site download faster. For example, instead of combined URL scripts look like this: MyPage.aspx?_TSM_HiddenField_'ctl00_ScriptManager_HiddenField and _TSM_CombinedScripts_ ... It will look like something like this: ... This would actually make a request for CDN instead of directly to your site because mycdnsubdomain.mysite.com is a domain displayed on CDN (of course not all CDN networks work that way, but most of them work in a way where you map a certain subdomin or other domain on CDN). This is a fairly quick and easy way to increase the download speed of your site with very little effort. Keep this in mind when you need to optimize the customer load side of things. Debug.Assert allows you to check different things during the debugging process or when you're assembling debugging. One of the problems with Debug.Assert is that it pops up the MessageBox dialogue. UI dialogues are a big no-no in server applications, as there are scenarios in which this UI dialogue will never show up, and your server application will just get stuck waiting for someone to press the dialogue buttons. One of the most common cases is when you don't see a dialogue when your server application is running under different credentials than yours is currently registered to the user. In this case, the UI dialogue will be hidden somewhere in different sessions with no chance of being noticed by any other user. .NET uses a variety of hooristics to figure out when to display to approve the dialogue. In most cases, in a server application, it won't show an approval dialogue at all (even when you're assembling debugging). As far as I could tell (and its now just calculated to guess) if you have agger installed and registered as a back-up, .NET will display the dialogue and that can create if your server app works under different credentials than is currently registered to the user. It seems there is a way to overcome this with a flag configuration. Add the following section to web.config (or app.config for apps not and it's going to disable the UI pop-up in the system. The application that makes it a dead end: If you have any trace of listeners that can pop up the user interface, you can also add the following configuration to the web.config, which will clear all traces of listeners: If you are a system. You're not sure that this is really your problem, it's very easy to figure it out with WinDbg, by joining the process, downloading SOS.dll and working with the team: q e!clrstack This team will show a managed stack of each thread in the application. so they can return the bug and you can safely ignore them). If you see a Call to Debug.Assert in the stack, and then MessageBox.Show, you can be sure that this is the problem you are facing. DataSets are a secure type wrapper around a dataset that reflects the structure of the database. It was created to make sure that the code accessing the database is a type of secure and any changes to the database structure that changes tables, columns, or column types will be caught during compilation, not during execution. If you have a large data set that contains a lot of tables, columns and relationships, it can be quite expensive to create it in terms of memory and time. The main reason why creating a large data set is expensive is because all the meta-data contained in data types sets (tables, columns, relationships) is created when you create a set of good data, even if you end up using it to get data from a single table. I can assume that the reason that all typed meta-data data is created during the instant processing of a set of data is because it inherits from the overall data set and access to meta data (tables, columns) can also be done in an unsafe form (i.e. access to table collection and/or table collection columns). If you're using a data set (or a dataset in general), you might be interested in the following tips: if you have a large data set, avoid it too many times in your app. This is especially painful for web applications, where every query can create a data set. Instead, you can use a shared dataset, but this can lead to errors due to changes in the database and the fact that you can only find these bugs during execution, not during compilation time (which basically misses the whole point of using the machine dataset in the first place). Datasets (datasets were entered into the kits) are inherited from the class This class implements IDisposable, which means that DataSets will actually be assembled after the end (you can read more about the completion and flow of the finalist here). To make sure that datasets are collected more often and not hagging around waiting for the final do do You call Recycling a dataset (or a set of imported data) or use a Use clause that will be called Recycling for you at the end of the code block. Don't use DataS kits at all ☹ consider using a different level of data access with a different approach, such as the one used in the SubSonic project. I think it would be pretty trivial creating a data set that is lazy in nature that creates meta-object data only when they are available for the first time. This will reduce the memory of a large data set, but will make the calculations used to create these objects a little less predictable. If you've before it or have already done a lazy data set pinging me through a contact form ☺ I've previously written about the internal .Net Framework Error Data Provider 6 and how to get a hot fix to fix the problem. It looks like this hot fix has never made it into .NET Framework 2.0 SP1 and there is a special fix (and KB article) to apply because the hotfix mentioned in the previous post just won't work. To get a hotfix to install .NET Framework 2.0 SP1, you need to request a fix for KB948815 in the hot web view patch request form. In relation to my previous post about the string. GetHashCode is used in AJAXControlToolkit in the ToolkitScriptManager class, I wanted to talk about the facility. GetHashCode in general, and the line. GetHashCode in particular. String. GetHashCode's documentation states: GetHashCode's behavior depends on its implementation, which can vary from one version of the total language run time to another. The reason why this can happen is to improve GetHashCode's performance. The real conclusion from this paragraph is that you don't have to base your implementation on a line. Gethashcode. But it's too harsh because GetHashCode in general and the strings. GetHashCode is specifically used throughout the running time for internal things. As long as you don't share this hash code outside of AppDomain borders it's relatively safe to use. The reason why you can't (or should say you shouldn't) pass it across the boundaries of AppDomain is that the main implemenetation object. GetHashCode must return an integer representing the object reference ID at .NET time. This link does not guarantee that it is the same for the same object in another application/process/machine. In the case of a line. GetHashCode, where implementation is different from the default implementation, you can pass it through AppDomains and even machines (although you don't have to count on it as well!) as long as they are in the same architecture, i.e., 32bit to 32bit and 64bit to 64bit. In general, the most recommended way to use x.GetHashCode simply does not use it at all. There are many implementations of hashing features built into .NET (such as MD5, SHA1, which are more consistent, but can be a little more expensive calculations. For all your external face code I would recommend using one of the common and known hashing hashing specifically, if this is the case where the code on the other hand should recompodo the hash and compare it with the hash passed. Published: Eran Sandler, at .NET, AJAX.NET, ASP.NET, Not Debugging Related Tags: .NET, 32bit, 64bit, ajaxcontroltoolkit, ASP.NET, GetHashCode, row. GetHashCode, toolkitscriptmanager If you mix and match 32 bit machines/processes with 64 bit machines/processes in a balanced load ASP.NET environment, and you use the class AJAXControlToolkit ToolkitManager, you may end up with the following error: Message: Build AjaxControlToolkit, Version No1.0.10920.32880, Culture-neutral, PublicToken-28f01b0e84b6d53e does not contain a script with a hashtag 9ea3f0e2. Stack trail: at AjaxControlToolkit.ToolkitScriptManager.DeserializeScriptEntries (StringizedScriptEntries, Boolean downloaded) at AjaxControlToolkit.ToolkitScriptManager.OnLoad in System.Web.UI.Control.LoadRecursive () on System.Web.UI.UI.Control.LoadRecursive Web.UI.Control.LoadRecursive () on System.Web.UI.Control.LoadRecursive () on System.UI.UI.Control.LoadRecursive () on System.UI.LoadRecursive () on System.Web.UI.Page.ProcessRequestMain (BooleanStaBeesForEsy). Boolean includeStagesAfterAsyncPoint) The reason is that the request sent to obtain combined javascripts management, used on the page, contains a hash code that is used internally. This hash code is generated from the script's name using the default line. GetHashCode feature, which returns different values for 32 bit and 64 bit processes. If your 64-bit process creates JavaScript include a call that contains 64 bits of hash code and the request will eventually reach the 32 bit process/machine you will get this error because the hash value will not be found internally and vice versa. There is an open problem about this in CodePlex since late January, but at the time of this post, the latest version of the source in CodePlex has no fixes for this. If you get an Internal .Net Error Framework Data Provider error 6 and you use S'L Server 2005 with a mirrored database know it is an error in managed customer code. Finish the review article 944099 KB for more information. At the time of writing, the only solution to this problem is a private hot fix that you need to get from Microsoft. Links all over the Internet to Hotfix Request web view forms seem to be broken. I contacted Microsoft and got this link replacement. Use this link to contact Microsoft, find out what you need hotfix for Article 944099 KB and submit a form. You will soon be contacted by a Microsoft representative who will provide you with detailed information about the download or request more information to better understand your needs. Posted by: Eran Sandler, at .NET, Debugging, SOS, Visual Studio, WinDbg Tags: .NET, . NET-Framework, Debugging, Managed Debugging, Managed Code, SOS, Sos.dll, Visual Studio, WinDbg I just asked via Ask the widget on on from this blog on Yedda (my day job) this question: Where is sos.dll for .net 3.5? I installed Visual Studio 2008, but I don't see sos.dll in the Microsoft.NET-Framework'v3.5 catalog? What's up with that? Topics: Debugging, managed code, debugging with cdb Asked dannyR January 03, 2008 View the entire discussion on Yedda, to which I replied: Where sos.dll for .net 3.5? There is a bit of confusion about the .NET Framework versions, unlike the CLR (Common Language Runtime) versions. Up to .NET Framework 2.0, the CLR versions have advanced side by side with the framework versions. Starting with .NET Framework v3.0, the CLR version is listed at 2.0 (there were no significant changes in the CLR itself), and the framework version continued to move forward. .NET Framework v3.5 actually runs on CLR 2.0 SP1. This means that you may be able to use SOS.dll located in Microsoft.NET-Framework-v2.0.50727 either with Visual Studio (as I explained in this post) or with WinDbg. Microsoft's Dave Broman wrote a post about displaying the .NET framework versions to the CLR version. Keep in mind that, as Dave states in his post, this is how he figures things in his head, not official documentation on the subject. Topics: Debugging, managed code, debugging with cdb Answered Eran on January 03, 2008 View the entire discussion on Yedda I know it's a bit confusing and I'm not quite sure why Microsoft has separated the framed version of the CLR versions (well, I can understand some thoughts leading up to this, but I don't think it was that good move), but a great Dave post makes things come to a head and lets find out. , in perspective, what version of CLR you actually use. Using. advanced .net debugging pdf download