Why is .NET exception not caught by try/catch block?

Asked 16 years, 3 months ago Modified 3 years, 1 month ago Viewed 77k times

56

I'm working on a project using the ANTLR parser library for C#. I've built a grammar to parse some text and it works well. However, when the parser comes across an illegal or unexpected token, it throws one of many exceptions. The problem is that in some cases (not all) that my try/catch block won't catch it and instead stops execution as an unhandled exception.

The issue for me is that I can't replicate this issue anywhere else but in my full code. The call stack shows that the exception definitely occurs within my try/catch(Exception) block. The only thing I can think of is that there are a few ANTLR assembly calls that occur between my code and the code throwing the exception and this library does not have debugging enabled, so I can't step through it. I wonder if non-debuggable assemblies inhibit exception bubbling? The call stack looks like this; external assembly calls are in Antlr.Runtime:

```
    Expl.Itinerary.dll!TimeDefLexer.mTokens() Line 1213 C#
    Antlr3.Runtime.dll!Antlr.Runtime.Lexer.NextToken() + 0xfc bytes
    Antlr3.Runtime.dll!Antlr.Runtime.CommonTokenStream.FillBuffer() + 0x22c
bytes
    Antlr3.Runtime.dll!Antlr.Runtime.CommonTokenStream.LT(int k = 1) + 0x68
bytes
    Expl.Itinerary.dll!TimeDefParser.prog() Line 109 + 0x17 bytes    C#
    Expl.Itinerary.dll!Expl.Itinerary.TDLParser.Parse(string Text = "",
Expl.Itinerary.IItinerary Itinerary = {Expl.Itinerary.MemoryItinerary})
Line 17 + 0xa bytes C#
```

The code snippet from the bottom-most call in Parse() looks like:

```
     try {
        // Execution stopped at parser.prog()
        TimeDefParser.prog_return prog_ret = parser.prog();
        return prog_ret == null ? null : prog_ret.value;
     }
     catch (Exception ex) {
        throw new ParserException(ex.Message, ex);
     }
```

To me, a catch (Exception) clause should've captured any exception whatsoever. Is there any reason why it wouldn't?

Update: I traced through the external assembly with Reflector and found no evidence of threading whatsoever. The assembly seems to just be a runtime utility class for

ANTLR's generated code. The exception thrown is from the TimeDefLexer.mTokens() method and its type is NoViableAltException, which derives from RecognitionException -> Exception. This exception is thrown when the lexer cannot understand the next token in the stream; in other words, invalid input. This exception is SUPPOSED to happen, however it should've been caught by my try/catch block.

Also, the rethrowing of ParserException is really irrelevant to this situation. That is a layer of abstraction that takes any exception during parse and convert to my own ParserException. The exception handling problem I'm experiencing is never reaching that line of code. In fact, I commented out the "throw new ParserException" portion and still received the same result.

One more thing, I modified the original try/catch block in question to instead catch NoViableAltException, eliminating any inheritance confusion. I still received the same result.

Someone once suggested that sometimes VS is overactive on catching handled exceptions when in debug mode, but this issue also happens in release mode.

Man, I'm still stumped! I hadn't mentioned it before, but I'm running VS 2008 and all my code is 3.5. The external assembly is 2.0. Also, some of my code subclasses a class in the 2.0 assembly. Could a version mismatch cause this issue?

Update 2: I was able to eliminate the .NET version conflict by porting relevant portions of my .NET 3.5 code to a .NET 2.0 project and replicate the same scenario. I was able to replicate the same unhandled exception when running consistently in .NET 2.0.

I learned that ANTLR has recently released 3.1. So, I upgraded from 3.0.1 and retried. It turns out the generated code is a little refactored, but the same unhandled exception occurs in my test cases.

Update 3: I've replicated this scenario in a simplified VS 2008 project. Feel free to download and inspect the project for yourself. I've applied all the great suggestions, but have not been able to overcome this obstacle yet.

If you can find a workaround, please do share your findings. Thanks again!

Thank you, but VS 2008 automatically breaks on unhandled exceptions. Also, I don't have a Debug->Exceptions dialog. The NoViableAltException that is thrown is fully intended, and designed to be caught by user code. Since it is not caught as expected, program execution halts unexpectedly as an unhandled exception.

The exception thrown is derived from Exception and there is no multi-threading going on with ANTLR.

c# .net exception antlr

Share

Improve this question

Follow

edited Mar 24, 2012 at 8:46

community wiki
9 revs, 4 users 88%
spoulson

25 Answers

Sorted by: Highest score (default)

32

I believe I understand the problem. The exception is being caught, the issue is confusion over the debugger's behavior and differences in the debugger settings among each person trying to repro it.

In the 3rd case from your repro I believe you are getting the following message: "NoViableAltException was unhandled by user code" and a callstack that looks like this:

```
         [External Code]
    >   TestAntlr-3.1.exe!TimeDefLexer.mTokens() Line 852 + 0xe bytes   C#
        [External Code]
        TestAntlr-3.1.exe!TimeDefParser.prog() Line 141 + 0x14 bytes    C#
        TestAntlr-3.1.exe!TestAntlr_3._1.Program.ParseTest(string Text =
"foobar;") Line 49 + 0x9 bytes C#
        TestAntlr-3.1.exe!TestAntlr_3._1.Program.Main(string[] args =
{string[0x00000000]}) Line 30 + 0xb bytes C#
        [External Code]
```

If you right click in the callstack window and run turn on show external code you see this:

```
        Antlr3.Runtime.dll!Antlr.Runtime.DFA.NoViableAlt(int s =
0x00000000, Antlr.Runtime.IIntStream input =
{Antlr.Runtime.ANTLRStringStream}) + 0x80 bytes

Antlr3.Runtime.dll!Antlr.Runtime.DFA.Predict(Antlr.Runtime.IIntStream input
= {Antlr.Runtime.ANTLRStringStream}) + 0x21e bytes
    >   TestAntlr-3.1.exe!TimeDefLexer.mTokens() Line 852 + 0xe bytes   C#
        Antlr3.Runtime.dll!Antlr.Runtime.Lexer.NextToken() + 0xc4 bytes
        Antlr3.Runtime.dll!Antlr.Runtime.CommonTokenStream.FillBuffer() +
0x147 bytes
        Antlr3.Runtime.dll!Antlr.Runtime.CommonTokenStream.LT(int k =
0x00000001) + 0x2d bytes
        TestAntlr-3.1.exe!TimeDefParser.prog() Line 141 + 0x14 bytes    C#
        TestAntlr-3.1.exe!TestAntlr_3._1.Program.ParseTest(string Text =
"foobar;") Line 49 + 0x9 bytes C#
        TestAntlr-3.1.exe!TestAntlr_3._1.Program.Main(string[] args =
{string[0x00000000]}) Line 30 + 0xb bytes C#
        [Native to Managed Transition]
        [Managed to Native Transition]
        mscorlib.dll!System.AppDomain.ExecuteAssembly(string assemblyFile,
System.Security.Policy.Evidence assemblySecurity, string[] args) + 0x39
bytes

Microsoft.VisualStudio.HostingProcess.Utilities.dll!Microsoft.VisualStudio.Hos
+ 0x2b bytes

mscorlib.dll!System.Threading.ThreadHelper.ThreadStart_Context(object
state) + 0x3b bytes

mscorlib.dll!System.Threading.ExecutionContext.Run(System.Threading.ExecutionC
executionContext, System.Threading.ContextCallback callback, object state)
+ 0x81 bytes
        mscorlib.dll!System.Threading.ThreadHelper.ThreadStart() + 0x40
bytes
```

The debugger's message is telling you that an exception originating outside your code (from NoViableAlt) is going through code you own in TestAntlr-3.1.exe!TimeDefLexer.mTokens() without being handled.

The wording is confusing, but it does not mean the exception is uncaught. The debugger is letting you know that code you own mTokens()" needs to be robust against this exception being thrown through it.

Things to play with to see how this looks for those who didn't repro the problem:

- Go to Tools/Options/Debugging and turn off "Enable Just My code (Managed only)". or option.
- Go to Debugger/Exceptions and turn off "User-unhandled" for Common-Language Runtime Exceptions.

Share

Improve this answer

Follow

edited Sep 5, 2008 at 5:54
Jeff Atwood
63.9k ● 48 ● 151 ● 153

answered Sep 3, 2008 at 5:27
Steve Steiner
5,339 ● 4 ● 33 ● 45

1 I was having a very similar problem - also with the ANTLR lexer that I had built for Silverlight... Turning off "Enable Just My Code" in the debugger solved the problem. It seems that the debugger was stepping in before my catch statement and if you didn't know what was going on it looked like the app was stopping without catching your exception. I don't fully understand why the debugger does this or how it distinguishes "my code" from "my libraries" etc. but this was the fix. (BTW, I'm not seeing the "user-unhandled" option in VS 2010. – Pat Niemeyer Oct 6, 2010 at 18:57

1 Once you disable "Enable Just My Code", the "User Unhandled" options go away from Debugger/Exceptions. The only option is checking "Thrown". – spoulson Jul 21, 2011 at 0:01

I can tell you what's happening here...

11

Visual Studio is breaking because it thinks the exception is unhandled. What does unhandled mean? Well, in Visual Studio, there is a setting in the Tools... Options... Debugging... General... "Enable Just My Code (Managed only)". If this is checked and if the exception propagates out of your code and out to a stack frame associated with a method call that exists in an assembly which is "NOT YOUR CODE" (for example, Antlr), that is considered "unhandled". I turn off that Enable Just My Code feature for this reason. But, if you ask me, this is lame... let's say you do this:

```
ExternalClassNotMyCode c = new ExternalClassNotMyCode();
try {
    c.doSomething( () => { throw new Exception(); } );
}
catch ( Exception ex ) {}
```

doSomething calls your anonymous function there and that function throws an exception...

Note that this is an "unhandled exception" according to Visual Studio if "Enable Just My Code" is on. Also, note that it stops as if it were a breakpoint when in debug mode, but in a non-debugging or production environment, the code is perfectly valid and works as expected. Also, if you just "continue" in the debugger, the app goes on it's merry way (it doesn't stop the thread). It is considered "unhandled" because the exception propagates through a stack frame that is NOT in your code (i.e. in the external library). If you ask me, this is lousy. Please change this default behavior Microsoft. This is a perfectly valid case of using Exceptions to control program logic. Sometimes, you can't change the third party library to behave any other way, and this is a very useful way to accomplish many tasks.

Take MyBatis for example, you can use this technique to stop processing records that are being collected by a call to SqlMapper.QueryWithRowDelegate.

Share
Improve this answer
Follow

edited Jan 20, 2012 at 19:39

community wiki
2 revs
Tony Schwartz

If you are looking to suggest a change in the way Visual Studio works, may I suggest Microsoft Connect? – Andrew Barber Jan 20, 2012 at 19:38

Primarily I am looking to explain the behavior, which was missing from this thread. The fact that I added commentary on the expected behavior is perfectly acceptable if you ask me. – Tony Schwartz Jan 20, 2012 at 19:42

I would debate that, but that wasn't the point of my comment. My comment was meant to *make a suggestion to you*, if you feel strongly about the issue. – Andrew Barber Jan 20, 2012 at 19:44

9

Regardless of whether the assembly has been compiled as a release build the exception should certainly 'bubble' up to the caller, there's no reason an assembly not being compiled in debug mode should have any affect on that.

I'd agree with Daniel is suggesting that perhaps the exception is occurring on a separate thread - try hooking the thread exception event in Application.ThreadException. This should be raised when any unhandled thread exception occurs. You could adapt your code thus:-

```
using System.Threading;

...

void Application_ThreadException(object sender, ThreadExceptionEventArgs e) {
  throw new ParserException(e.Exception.Message, e.Exception);
}

 ...

 var exceptionHandler =
    new ThreadExceptionEventHandler(Application_ThreadException);
 Application.ThreadException += exceptionHandler;
 try {
    // Execution stopped at parser.prog()
    TimeDefParser.prog_return prog_ret = parser.prog();
    return prog_ret == null ? null : prog_ret.value;
 }
 catch (Exception ex) {
    throw new ParserException(ex.Message, ex);
 }
 finally {
    Application.ThreadException -= exceptionHandler;
 }
```

Share Improve this answer Follow

answered Aug 30, 2008 at 15:34

ljs
37.8k ● 36 ● 109 ● 124

5

Are you using .Net 1.0 or 1.1? If so then catch(Exception ex) won't catch exceptions from unmanaged code. You'll need to use catch {} instead. See this article for further details:

http://www.netfxharmonics.com/2005/10/net-20-trycatch-and-trycatchexception/

Share Improve this answer Follow

answered Aug 30, 2008 at 16:07

tbreffni
5,132 ● 5 ● 32 ● 30

4 Would have been great if you would have summarized the link, because the article is no longer at that address. – pqsk Aug 6, 2014 at 15:11

1 web.archive.org/web/20080723164904/http://… – Luis Sep 22, 2017 at 4:41

4

I'm with @Shaun Austin - try wrapping the try with the fully qualified name

```
catch (System.Exception)
```

and see if that helps.Does the ANTLR doc say what Exceptions should be thrown?

Share Improve this answer Follow

answered Sep 2, 2008 at 17:08

JamesSugrue
15k ● 10 ● 62 ● 93

3

Is it possible that the exception is being thrown in another thread? Obviously your calling code is single threaded, but maybe the library you are consuming is doing some multithreaded operations under the covers.

Share
Improve this answer
Follow

edited Aug 30, 2008 at 15:36

answered Aug 30, 2008 at 15:10

Daniel Auger
12.6k ● 5 ● 53 ● 73

2

> To me, a catch (Exception) clause should've captured any exception whatsoever. Is there any reason why it wouldn't?

The only possibility I can think of is that something else is catching it before you and handling it in a way that appears to be an uncaught exception (e.g. exiting the process).

> my try/catch block won't catch it and instead stops execution as an unhandled exception.

You need to find what is causing the exit process. It might be something other than an unhandled exception. You might try using the native debugger with a breakpoint set on "{,,kernel32.dll}ExitProcess". Then use SOS to determine what managed code is calling exit process.

Share Improve this answer Follow

answered Sep 2, 2008 at 4:28
Steve Steiner
5,339 ● 4 ● 33 ● 45

2

Personally I'm not convinced by the threading theory at all.

The one time I've seen this before, I was working with a library which also defined Exception and the usings I had meant that the actual Catch was referring to a different "Exception" type (if it had been fully qualified it was Company.Lib.Exception but it wasnt because of the using) so when it came to catching a normal exception that was being thrown (some kind of argument exception if I remember correctly) it just wouldn't catch it because the type didn't match.

So in summary, is there another Exception type in a different namespace that is in a using in that class?

EDIT: A quick way to check this is make sure in your catch clause you fully qualify the Exception type as "System.Exception" and give it a whirl!

EDIT2: OK I've tried the code and concede defeat for now. I'll have to have another look at it in the morning if no one has come up with a solution.

Share
Improve this answer
Follow

edited Sep 2, 2008 at 20:33

answered Sep 2, 2008 at 16:00
Shaun Austin
3,822 ● 3 ● 24 ● 27

2

Hmm, I don't understand the problem. I downloaded and tried your example solution file.

An exception is thrown in TimeDefLexer.cs, line 852, which is subsequently handled by the catch block in Program.cs that just says *Handled exception*.

If I uncomment the catch block above it, it will enter that block instead.

What seems to be the problem here?

As Kibbee said, Visual Studio will stop on exceptions, but if you ask it to continue, the exception will get caught by your code.

Share Improve this answer Follow

answered Sep 2, 2008 at 20:47
Lasse V. Karlsen
391k ● 106 ● 646 ● 844

2

I downloaded the sample VS2008 project, and am a bit stumped here too. I was able to get past the exceptions however, although probably not in a way that will work will great for you. But here's what I found:

This mailing list post had a discussion of what looks to be the same issue you are experiencing.

From there, I added a couple dummy classes in the main program.cs file:

```
class MyNoViableAltException : Exception
{
    public MyNoViableAltException()
    {
    }
    public MyNoViableAltException(string grammarDecisionDescription, int 
decisionNumber, int stateNumber, Antlr.Runtime.IIntStream input)
    {
    }
}
class MyEarlyExitException : Exception
{
    public MyEarlyExitException()
    {
    }

    public MyEarlyExitException(int decisionNumber, Antlr.Runtime.IIntStream 
input)
    {
    }
}
```

and then added the using lines into TimeDefParser.cs and TimeDefLexer.cs:

```
using NoViableAltException = MyNoViableAltException;
using EarlyExitException = NoViableAltException;
```

With that the exceptions would bubble into the fake exception classes and could be handled there, but there was still an exception being thrown in the mTokens method in TimeDefLexer.cs. Wrapping that in a try catch in that class caught the exception:

```
            try
            {
                alt4 = dfa4.Predict(input);
            }
            catch
            {
            }
```

I really don't get why wrapping it in the internal method rather than where it is being called from handle the error if threading isn't in play, but anyways hopefully that will point someone smarter than me here in the right direction.

Share Improve this answer Follow

answered Sep 2, 2008 at 22:27
Scott Nichols
6,228 ● 4 ● 30 ● 23

2

I downloaded your code and everything work as expected.

Visual Studio debugger correctly intercepts all exceptions. Catch blocks work as expected.

I'm running Windows 2003 server SP2, VS2008 Team Suite (9.0.30729.1 SP)

I tried to compile you project for .NET 2.0, 3.0 & 3.5

@Steve Steiner, debugger options you mentioned have nothing to do with this behavior.

I tried to play with these options with no visible effects - catch blocks managed to intercept all exceptions.

Share
Improve this answer
Follow

edited Sep 3, 2008 at 5:52

answered Sep 3, 2008 at 5:44
aku
124k ● 33 ● 176 ● 203

2

Steve Steiner is correct that the exception is originating in the antlr library, passing through the mTokens() method and being caught in the antlr library. The problem is that this method is auto-generated by antlr. Therefore, any changes to handle the exception in mTokens() will be overwritten when your generate your parser/lexer classes.

By default, antlr will log errors and try to recover parsing. You can override this so that the parser.prog() will throw an exception whenever an error is encountered. From your example code i think this is the behaviour you were expecting.

Add this code to your grammer (.g) file. You will also need to turn off "Enable Just My Code" in the debugging menu.

```
@members {

    public override Object RecoverFromMismatchedSet(IIntStream
input,RecognitionException e,    BitSet follow)
    {
        throw e;
    }
}
```

```
@rulecatch {
    catch (RecognitionException e)
    {
        throw e;
    }
}
```

This is my attempt at a C# version of the example given in the "Exiting the recogniser on first error" chapter of the "Definitive ANTLR Reference" book.

Hope this is what you were looking for.

Share
Improve this answer
Follow

answered Mar 10, 2009 at 12:57

community wiki
David Turvey

2

You can set up VS.Net to break as soon as any exception occurs. Just run your project in debug mode, and it will stop as soon as the exception is thrown. Then you should have a better idea of why it isn't being caught.

Also, you can put some code in to catch all unhandled exceptions.

```
Application.ThreadException += new
ThreadExceptionEventHandler(ThreadExceptionHandler);

 // Catch all unhandled exceptions in all threads.
 AppDomain.CurrentDomain.UnhandledException += new
UnhandledExceptionEventHandler(UnhandledExceptionHandler);
```

Share
Improve this answer
Follow

edited Nov 18, 2021 at 9:45
Zoe - Save the data dump ♦
28.1k ● 22 ● 127 ● 158

answered Aug 30, 2008 at 15:36
Kibbee
66.1k ● 28 ● 144 ● 184

1

Oh and in reference to what Kibbee said; if you select Debug|Exceptions in VS and just click all the boxes in the 'thrown' column it should pick *everything* up AFAIK as a 'first chance exception', i.e. VS will indicate when the exception is *about* to be processed by everything else and break on the relevant code. This should help with debugging.

Share Improve this answer Follow

answered Aug 30, 2008 at 15:42
ljs
37.8k ● 36 ● 109 ● 124

1

The best option sounds like setting Visual Studio to break on all unhandled exceptions (Debug -> Exceptions dialog, check the box for "Common Language Runtime Exceptions" and possibly the others as well). Then run your program in debug mode. When the ANTLR parser code throws an exception it should be caught by Visual Studio and allow you to see where it is occurring, the exception type, etc.

Based on the description, the catch block appears to be correct, so one of several things could be happening:

1. the parser is not actually throwing an exception
2. the parser is ultimately throwing something that isn't deriving from System.Exception
3. there is an exception being thrown on another thread that isn't being handled

It sounds like you have potentially ruled out issue #3.

Share Improve this answer Follow

answered Aug 31, 2008 at 14:58

Scott Dorman
42.5k ● 12 ● 81 ● 112

1

> I traced through the external assembly with Reflector and found no evidence of threading whatsoever.

You can't find any threading does not mean there is no threading

.NET has a 'thread pool' which is a set of 'spare' threads that sit around mostly idle. Certain methods cause things to run in one of the thread pool threads so they don't block your main app.

The blatant examples are things like ThreadPool.QueueUserWorkItem, but there are lots and lots of other things which can also run things in the thread pool that don't look so obvious, like Delegate.BeginInvoke

Really, you need to do what kibbee suggests.

Share
Improve this answer
Follow

edited May 23, 2017 at 12:00
Community Bot
1 ● 1

answered Sep 1, 2008 at 4:02
Orion Edwards
123k ● 66 ● 245 ● 339

have you tried to print (Console.WriteLine()) the exception inside the catch clause, and not use visual studio and run your application on console?

1

Share Improve this answer Follow

answered Sep 2, 2008 at 15:45

Eduardo Diaz
241 ●1 ●3 ●6

1

I believe Steve Steiner is correct. When researching Steve's suggestions, I came across this thread talking about the "Enable Just My Code" option in Tools|Options|Debugger|General. It is suggested that the debugger will break in certain conditions when non-user code either throws or handles an exception. I'm not exactly sure why this even matters, or why the debugger specifically says the exception was unhandled when it really was.

I was able to eliminate the false breaks by disabling the "Enable Just My Code" option. This also changes the Debug|Exceptions dialog by removing the "User-handled" column as it no longer applies. Or, you can just uncheck the "User-handled" box for CLR and get the same result.

Bigtime thanks for the help everyone!

Share Improve this answer Follow

answered Sep 3, 2008 at 10:17

spoulson
21.6k ●16 ●78 ●102

0

> "Also, you can put some code in to catch all unhandled exceptions. Read the link for more info, but the basics are these two lines."

This is false. This used to catch all unhandled exceptions in .NET 1.0/1.1 but it was a bug and it wasn't supposed to and it was fixed in .NET 2.0.

```
AppDomain.CurrentDomain.UnhandledException
```

Is only intended to be used as a last chance logging saloon so you can log the exception before the program exits. It wont catch the exception as of 2.0 onwards (although in .NET 2.0 at least there is a config value you can modify to make it act like 1.1 but it isn't recommended practice to use this.).

Its worth noting that there are few exceptions that you *cannot* catch, such as StackOverflowException and OutOfMemoryException. Otherwise as other people have suggested it might be an exception in a background thread somewhere. Also I'm pretty sure you can't catch some/all unmanaged/native exceptions either.

Share
Improve this answer
Follow

edited Aug 30, 2008 at 16:10

answered Aug 30, 2008 at 16:04
Quibblesome
25.4k ● 10 ● 62 ● 104

0

I don't get it...your catch block just throws a new exception (with the same message). Meaning that your statement of:

> The problem is that in some cases (not all) that my try/catch block won't catch it and instead stops execution as an unhandled exception.

is exactly what is *expected* to happen.

Share Improve this answer Follow

answered Aug 30, 2008 at 16:10
Mark Brackett
85.6k ● 17 ● 111 ● 155

0

I agree with Daniel Auger and kronoz that this smells like an exception that has something to do with threads. Beyond that, here are my other questions:

1. What does the complete error message say? What kind of exception is it?
2. Based on the stack trace you've provided here, isn't the exception thrown by you code in TimeDefLexer.mTokens()?

Share
Improve this answer
Follow

edited May 23, 2017 at 11:46
Community Bot
1 ● 1

answered Aug 30, 2008 at 16:19
flipdoubt
14.4k ● 16 ● 66 ● 98

0

I'm not sure if I'm being unclear, but if so, I'm seeing the debugger halt execution with an "Unhandled Exception" of type NoViableAltException. Initially, I didn't know anything about this Debug->Exceptions menu item because MS expects you, at VS install time, to commit to a profile when you have no idea how they are different. Apparently, I was not on the C# dev profile and was missing this option. After finally debugging all thrown CLR exceptions, I was unfortunately unable to discover any new behavior leading to the reason for this unhandled exception issue. All the exceptions thrown were expected and supposedly handled in a try/catch block.

I reviewed the external assembly and there is no evidence of multithreading. By that, I mean no reference exists to System.Threading and no delegates were used whatsoever. I'm familiar with that constitutes instantiating a thread. I verify this by observing the Threads toolbox at the time of the unhandled exception to view there is only one running thread.

I have an open issue with the ANTLR folks so perhaps they've been able to tackle this issue before. I've been able to replicate it in a simple console app project using .NET 2.0 and 3.5 under VS 2008 and VS 2005.

It's just a pain point because it forces my code to only work with known valid parser input. Using an `IsValid()` method would be risky if it threw an unhandled exception based on user input. I'll keep this question up to date when more is learned of this issue.

Share Improve this answer Follow

answered Sep 2, 2008 at 14:08
spoulson
21.6k ● 16 ● 78 ● 102

0

@spoulson,

If you can replicate it, can you post it somewhere? One avenue you could try is usign WinDBG with the SOS extensions to run the app and catch the unhandled exception. It will break on the first chance exception (before the runtime tries to find a handler) and you can see at that point where it is coming from, and what thread.

If you haven't used WinDBG before, it can be a little overwhelming, but here's a good tutorial:

http://blogs.msdn.com/johan/archive/2007/11/13/getting-started-with-windbg-part-i.aspx

Once you start up WinDBG, you can toggle the breaking of unhandled exceptions by going to Debug->Event Filters.

Share Improve this answer Follow

answered Sep 2, 2008 at 14:30
Cory Foy
7,212 ● 4 ● 32 ● 34

0

Wow, so of the reports so far, 2 worked correctly, and 1 experienced the issue I reported. What are the versions of Windows, Visual Studio used and .NET framework with build numbers?

I'm running XP SP2, VS 2008 Team Suite (9.0.30729.1 SP), C# 2008 (91899-270-92311015-60837), and .NET 3.5 SP1.

Share Improve this answer Follow

answered Sep 3, 2008 at 1:21

spoulson
21.6k ● 16 ● 78 ● 102

0

If you are using com objects your project and try catch blocks not catch the exceptions you will be need disable Tools/Debugging/Break when exceptions cross AppDomain or managed/native boundaries(Managed only) option.

Share

Improve this answer

Follow

answered Feb 17, 2017 at 9:19

community wiki
Daghan Karakasoglu