XmlSerializer giving FileNot

Asked 14 years, 8 months ago Modified 11 months a

398

An application I've been working with is

A statement like

```
XmlSerializer lizer = new XmlSerializ
```

produces:

```
System.IO.FileNotFoundException occurred
  Message="Could not load file or assembly '[Containing Assembly of
MyType].XmlSerializers, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null' or
one of its dependencies. The system cannot find the file specified."
  Source="mscorlib"
  FileName="[Containing Assembly of MyType].XmlSerializers, Version=1.0.0.0,
Culture=neutral, PublicKeyToken=null"
  FusionLog=""
  StackTrace:
       at System.Reflection.Assembly._nLoad(AssemblyName fileName, String
codeBase, Evidence assemblySecurity, Assembly locationHint, StackCrawlMark&
stackMark, Boolean throwOnFileNotFound, Boolean forIntrospection)
       at System.Reflection.Assembly.nLoad(AssemblyName fileName, String
codeBase, Evidence assemblySecurity, Assembly locationHint, StackCrawlMark&
stackMark, Boolean throwOnFileNotFound, Boolean forIntrospection)
```

I don't define any special serializers for my class.

How can I fix this problem?

c# xml-serialization

Share Improve this question Follow

edited Nov 13, 2013 at 18:16
Peter Mortensen
31k 22 108 132

asked Jul 14, 2009 at 19:19
Irwin
12.7k 11 68 97

5 OK, so this question is just my C# version of an already asked VB question: stackoverflow.com/questions/294659/... Thanks guys. – Irwin Jul 14, 2009 at 19:51

2 Six years on, @VladV 's answer is the simplest and the least adverse-affecting solution. Just change the `Generate serialization assembly` drop-down to "On", instead of "Auto". – Riegardt Steyn Jul 23, 2015 at 12:02

@Heliac: I disagree. It does not always work. Please see Benoit Blanchon's comment to Vlad's answer. The simplest answer for me is to not use String.Collection in config files. Instead I use: string[] items = Settings.Default.StringofNewlineDelimitedItems.Split(new[] {Environment.NewLine}); – Andrew Dennison Dec 17, 2015 at 19:44

21 Answers

Sorted by: Highest score (default)

455

Believe it or not, this is normal behaviour. An exception is thrown but handled by the XmlSerializer, so if you just ignore it everything should continue on fine.

I have found this very annoying, and there have been many complaints about this if you search around a bit, but from what I've read Microsoft don't plan on doing anything about it.

You can avoid getting Exception popups all the time while debugging if you switch off first chance exceptions for that specific exception. In Visual Studio, go to *Debug* -> *Exceptions* (or press Ctrl + Alt + E), *Common Language Runtime Exceptions* -> *System.IO* -> *System.IO.FileNotFoundException*.

You can find information about another way around it in the blog post *C# XmlSerializer FileNotFound exception* (which discusses Chris Sells' tool *XmlSerializerPreCompiler*).

Share Improve this answer Follow

edited Aug 31, 2021 at 9:48
Moon Waxing
695 14 20

answered Jul 24, 2009 at 11:21

Martin Sherburn
6,048 2 23 17

180 One of the possible ways to get rid of this problem is check "Just my code" option in Tools -> Options -> Debugging -> General options. – Frederic Mar 18, 2010 at 10:42

28 @Frederic: This comment is awesome! I'm sitting here with a "WTF!?" expression on my face, trying to hunt this spurious exception down, and I find this question, with answer (It's Microsoft's fault, what else is new?), but I didn't want to disable exception handling, because I might need it for my code. A+! – Kumba Jan 4, 2011 at 1:59

31 I think Hans' suggestion below is more valuable - use a different method call that does not produce this exception at all: XmlSerializer serializer = XmlSerializer.FromTypes(new[] { typeof(MyType) })[0]; – bright Jun 9, 2012 at 7:47

3 The problem is that this fails my test, so I cannot just "ignore" the exception – Csaba Toth Jun 28, 2013 at 16:08

20 I'm sorry, but this is a terrible suggestion. FileNotFoundException is one of the more common ones, in my experience, and disabling this exception reporting is just asking for trouble someday in the future. Better to turn on 'Just My Code' or enable the creation of serialization assemblies described below. – Quark Soup Nov 22, 2013 at 15:44

Like Martin Sherburn said, this is normal behavior. The constructor of the XmlSerializer first tries to find an assembly named [YourAssembly].XmlSerializers.dll which should contain the generated

116

class for serialization of your type. Since such a DLL has not been generated yet (they are not by default), a FileNotFoundException is thrown. When that happenes, XmlSerializer's constructor catches that exception, and the DLL is generated automatically at runtime by the XmlSerializer's constructor (this is done by generating C# source files in the %temp% directory of your computer, then compiling them using the C# compiler). Additional constructions of an XmlSerializer for the same type will just use the already generated DLL.

> **UPDATE:** Starting from .NET 4.5, `XmlSerializer` no longer performs code generation nor does it perform compilation with the C# compiler in order to create a serializer assembly at runtime, unless explicitly forced to by setting a configuration file setting (useLegacySerializerGeneration). This change removes the dependency on `csc.exe` and improves startup performance. *Source: .NET Framework 4.5 Readme, section 1.3.8.1.*

The exception is handled by XmlSerializer's constructor. There is no need to do anything yourself, you can just click 'Continue' (F5) to continue executing your program and everything will be fine. If you're bothered by the exceptions stopping the execution of your program and popping up an exception helper, you either have 'Just My Code' turned off, or you have the FileNotFoundException set to break execution when thrown, instead of when 'User-unhandled'.

To enable 'Just My Code', go to Tools >> Options >> Debugging >> General >> Enable Just My Code. To turn off breaking of execution when FileNotFound is thrown, go to Debug >> Exceptions >> Find >> enter 'FileNotFoundException' >> untick the 'Thrown' checkbox from System.IO.FileNotFoundException.

Share Improve this answer Follow

edited Oct 31, 2012 at 18:32

answered Aug 3, 2009 at 19:51

Allon Guralnek
16k 6 62 94

+1 for the update: this explains the different behavior when debugging test cases – mbx Sep 19, 2014 at 7:04

3 Your update suggests that this exception should not occur in .NET 4.5, but I am still seeing it. – Tim Sparkles Feb 20, 2015 at 19:44

@Timbo: I don't see why you wouldn't get that exception with .NET 4.5. It still looks for a file, and if the file is missing, a `FileNotFoundException` will be thrown. The difference is not in how the assembly's existence is checked, but in how to generate it once it's determined that it's missing. Before, it used textual C# code generation with a call to the C# compiler to create the IL. Starting with .NET 4.5 it emits IL directly, without the use of a compiler. – Allon Guralnek Feb 23, 2015 at 13:58

2 I just wish MS would implement this as if (File.Exists(...)) { Load } else { Fallback } instead of try { Load } catch { Fallback }. Exception-based flow control smells bad and makes my debugging experience more difficult and brittle than necessary. – Tim Sparkles Jul 28, 2015 at 22:19

1 @Timbo: A simple `File.Exists()` may not be sufficient. Locating an assembly is not a simple affair, the runtime looks in several locations and I believe the behaviour changes according to the environment (console application vs being hosted in IIS, etc). I guess what should have been implemented was a `TryLoadAssembly()` or something similar. – Allon Guralnek Jul 29, 2015 at 4:40

72

In Visual Studio project properties ("Build" page, if I recall it right) there is an option saying "generate serialization assembly". Try turning it on for a project that generates *[Containing Assembly of MyType]*.

Share Improve this answer Follow

answered Jul 20, 2009 at 20:28

VladV
10.2k 3 34 48

4 Also see stackoverflow.com/a/8798289/1164966 if the serialization assembly is still not generated by Visual Studio. – Benoit Blanchon Apr 22, 2014 at 12:44

1 Best, clearest, succinct answer! I wish I could vote up again too! – John Zabroski Oct 7, 2016 at 22:05

67

There is a workaround for that. If you use

```
XmlSerializer lizer = XmlSerializer.FromTypes(new[] { typeof(MyType) })[0];
```

it should avoid that exception. This worked for me.

WARNING: *Do not use multiple times, or you will have a **memory leak***

You will leak memory like crazy if you use this method to create instances of `XmlSerializer` for the same type more than once!

This is because this method bypasses the built-in caching provided the `XmlSerializer(type)` and `XmlSerializer(type, defaultNameSpace)` constructors (all other constructors also bypass the cache).

If you use any method to create an XmlSerializer that is not via these two constructors, you must implement your own caching or you'll hemorrhage memory.

Share Improve this answer Follow

edited Mar 22, 2017 at 21:58

Tom Leys
18.7k 7 41 63

answered Mar 30, 2012 at 13:25

quadfinity
887 1 8 9

45 **WARNING:** You will leak memory like crazy if you use this method to create instances of `XmlSerializer` for the same type more than once! This is because this method bypasses the built-in caching provided the

`XmlSerializer(type)` and `XmlSerializer(type, defaultNameSpace)` constructors (all other constructors also bypass the cache). If you use any method to create an `XmlSerializer` that is not via these two constructors, you must implement your own caching or you'll hemorrhage memory. – Allon Guralnek Oct 31, 2012 at 18:10

4 @AllonGuralnek Well I'll be damned...you are absolutely correct; further digging in via Reflector shows that while it does check the cache, it does so **after** generating the serialization assembly! Wtf?!? – JerKimball Jan 25, 2013 at 20:02

4 Turns out its a known bug: weblogs.asp.net/cschittko/archive/2005/01/14/353435.aspx – JerKimball Jan 25, 2013 at 20:22

3 @JerKimball: That page isn't actually lying. As you discovered, `FromTypes` does appear to populate the cache. So it should be a valid way to warm up an empty `XmlSerializer` cache in one statement (like the article suggests), but a really bad way to retrieve anything from it (should only be done via the simplest constructors). In any case, I didn't know it was a bug, I always thought anything that leaks is supposed to leak (like the more advanced `XmlSerializer` constructors). I wouldn't have even considered using `FromTypes()` since you can just do `types.Select(t => new XmlSerializer(t))` . – Allon Guralnek Jan 25, 2013 at 23:14

2 @AllonGuralnek The non-probing aspect of using `FromTypes` does have its appeal - even tho the exceptions thrown are all caught, it is a pricely operation; the 'cache your own way' approach appears to be the only workaround, as the only officially supported fix looks to be in an obscure web-based assembly. (edit: frankly, I'm all for porting everything over to data contracts :)) – JerKimball Jan 25, 2013 at 23:39

31

I ran into this exact issue and couldn't get around it by any of the solutions mentioned.

Then I finally found a solution. It appears that the serializer needs not only the type, but the nested types as well. Changing this:

```
XmlSerializer xmlSerializer = new XmlSerializer(typeof(T));
```

To this:

```
XmlSerializer xmlSerializer = new XmlSerializer(typeof(T).GetNestedTypes());
```

Fixed the issue for me. No more exceptions or anything.

Share Improve this answer Follow

edited Mar 27, 2018 at 7:27

stema
91.6k 20 109 135

answered Sep 15, 2016 at 14:06

Frosty
321 3 2

14 This worked for me. Using .Net4.0 the format is `var xmlSerializer = new XmlSerializer(typeof(T), typeof(T).GetNestedTypes());` – user3161729 Sep 26, 2016 at 13:29

1 This worked for me as well. But it only seems to be necessary when serializing, not when deserializing. Maybe that makes sense, maybe it doesn't. – SteveCinq Sep 3, 2018 at 1:03

2 This also produces memory leak, if run a lot of times. – Volodymyr Kotylo Mar 13, 2019 at 6:58

This worked for me using .NET7 and Visual Studio 2022. – rrirower May 13, 2023 at 14:03

9

My solution is to go straight to reflection to create the serializer. This bypasses the strange file loading that causes the exception. I packaged this in a helper function that also takes care of caching the serializer.

```
private static readonly Dictionary<Type,XmlSerializer> _xmlSerializerCache = new
Dictionary<Type, XmlSerializer>();

public static XmlSerializer CreateDefaultXmlSerializer(Type type)
{
    XmlSerializer serializer;
    if (_xmlSerializerCache.TryGetValue(type, out serializer))
    {
        return serializer;
    }
    else
    {
        var importer = new XmlReflectionImporter();
        var mapping = importer.ImportTypeMapping(type, null, null);
        serializer = new XmlSerializer(mapping);
        return _xmlSerializerCache[type] = serializer;
    }
}
```

Share Improve this answer Follow

answered Feb 23, 2012 at 16:16

d--b
5,699 3 42 69

2 problems here - first your code isn't thread-safe, and second (more importantly) you are attempting to replicate what the .net runtime already does (based on the ctor you are using). i.e. there is no need for this code – Dave Black Jul 28, 2015 at 19:42

@DaveBlack: Yes, quadfinity's answer with caching to a ConcurrentDictionary would be better – d--b Jul 30, 2015 at 13:52

@d-b My 2nd point was that caching is not even needed - as long as you are using one of the 2 ctors that the framework caches (OP is using the first). From MSDN: To increase performance, the XML serialization infrastructure dynamically generates assemblies to serialize and deserialize specified types. The framework finds and reuses those assemblies. This behavior occurs only when using the following ctors: XmlSerializer.XmlSerializer(Type) XmlSerializer.XmlSerializer(Type, String) Reference: msdn.microsoft.com/en-us/library/... – Dave Black Jul 30, 2015 at 15:16

@DaveBlack: Yes, but these constructors throw and catch an exception internally even when the usage is completely valid. This is bad, and this is the reason why the OP asked the question in the first place. – d--b Jul 31, 2015 at 8:47

@d-b True, but what I meant to say (but wasn't clear - my apologies) was that the only lines of your soln that are necessary are the first 3 lines in the else condition. – Dave Black Jul 31, 2015 at 14:54

8

To avoid the exception you need to do two things:

1. Add an attribute to the serialized class (I hope you have access)
2. Generate the serialization file with sgen.exe

Add the System.Xml.Serialization.XmlSerializerAssembly attribute to your class. Replace 'MyAssembly' with the name of the assembly where MyClass is in.

```
[Serializable]
[XmlSerializerAssembly("MyAssembly.XmlSerializers")]
public class MyClass
{
…
}
```

Generate the serialization file using the sgen.exe utility and deploy it with the class's assembly.

'sgen.exe MyAssembly.dll' will generate the file MyAssembly.XmlSerializers.dll

These two changes will cause the .net to directly find the assembly. I checked it and it works on .NET framework 3.5 with Visual Studio 2008

Share Improve this answer Follow edited Oct 29, 2009 at 12:53 answered Oct 29, 2009 at 11:01

Ami Bar
81 1 2

Ok, and did it fail without these changes, and if so, why? – John Saunders Oct 30, 2009 at 1:51

1 I can find no reason for why my project, 4.0 in VS2012, suddenly started failing. "Ignoring" the error was not an option, because it occurred every time I tried to access Active Directory; thus ignoring would mean not authenticating. I am still very frustrated that VS2012 won't auto-generate the serialization DLL properly. However, these steps provided the perfect solution. – sfuqua Apr 2, 2013 at 13:10

7

Function XmlSerializer.FromTypes does not throw the exception, but it leaks the memory. Thats why you need to cache such serializer for every type to avoid memory leaking for every instance created.

Create your own XmlSerializer factory and use it simply:

```
XmlSerializer serializer = XmlSerializerFactoryNoThrow.Create(typeof(MyType));
```

The factory looks likes:

```
public static class XmlSerializerFactoryNoThrow
{
```

```
    public static Dictionary<Type, XmlSerializer> _cache = new Dictionary<Type,
XmlSerializer>();

    private static object SyncRootCache = new object();

    /// <summary>
    /// //the constructor XmlSerializer.FromTypes does not throw exception, but
it is said that it causes memory leaks
    /// http://stackoverflow.com/questions/1127431/xmlserializer-giving-
filenotfoundexception-at-constructor
    /// That is why I use dictionary to cache the serializers my self.
    /// </summary>
    public static XmlSerializer Create(Type type)
    {
        XmlSerializer serializer;

        lock (SyncRootCache)
        {
            if (_cache.TryGetValue(type, out serializer))
                return serializer;
        }

        lock (type) //multiple variable of type of one type is same instance
        {
            //constructor XmlSerializer.FromTypes does not throw the first chance
exception
            serializer = XmlSerializer.FromTypes(new[] { type })[0];
            //serializer = XmlSerializerFactoryNoThrow.Create(type);
        }

        lock (SyncRootCache)
        {
            _cache[type] = serializer;
        }
        return serializer;
    }
}
```

More complicated version without possibility of memory leak (please someone review the code):

```
    public static XmlSerializer Create(Type type)
    {
        XmlSerializer serializer;

        lock (SyncRootCache)
        {
            if (_cache.TryGetValue(type, out serializer))
                return serializer;
        }

        lock (type) //multiple variable of type of one type is same instance
        {
            lock (SyncRootCache)
            {
                if (_cache.TryGetValue(type, out serializer))
                    return serializer;
            }
            serializer = XmlSerializer.FromTypes(new[] { type })[0];
            lock (SyncRootCache)
            {
```

```
                    _cache[type] = serializer;
                }
            }
            return serializer;
        }
    }
```

Share Improve this answer Follow

edited Nov 23, 2017 at 15:39

answered Sep 22, 2016 at 15:14

Tomas Kubes
24.5k 18 116 157

You should use ConcurrentDictionary instead. This code can deadlock. – Behrooz Nov 21, 2017 at 12:43

1 Sorry, I got the words confused. What I meant is that it can Insert an item more than once. because there is a gap between when it checks for existance and when it inserts. concurrent dictionary uses some kind of two-phase locking (bag[0] and then bag[hash]]) and keeps a reference to the bag that must insert/contain the item you're working. It's faster, safer and cleaner. – Behrooz Nov 23, 2017 at 12:57

1 Yes and no. You are right that it can happen that in the same time one serializer of same type will be created on two threads in parallel and then added to dictionary twice. In such case second insert will just replace the first one, but the lock section guarantee thread safety and the overall disadvantage is small memory leak. This is performance optimization, because you don't want thread one with Serializer of type A wait be blocked by thread two with serializer of type B in real scenario. – Tomas Kubes Nov 23, 2017 at 15:32

1 @Behrooz Please check the new version of source code. – Tomas Kubes Nov 23, 2017 at 15:39

1 The 2nd version looks good to me. btw, you can use _cache itself for locking. – Behrooz Nov 24, 2017 at 22:52

6

This exception can also be trapped by a managed debugging assistant (MDA) called BindingFailure.

This MDA is useful if your application is designed to ship with pre-build serialization assemblies. We do this to increase performance for our application. It allows us to make sure that the pre-built serialization assemblies are being properly built by our build process, and loaded by the application without being re-built on the fly.

It's really not useful except in this scenario, because as other posters have said, when a binding error is trapped by the Serializer constructor, the serialization assembly is re-built at runtime. So you can usually turn it off.

Share Improve this answer Follow

edited Nov 13, 2013 at 19:26

Peter Mortensen
31k 22 108 132

answered Aug 7, 2009 at 0:15

HiredMind
1,837 17 27

In Visual Studio project properties there is an option saying "generate serialization assembly". Try turning it on for a project that generates [Containing Assembly of MyType].

2

Share Improve this answer Follow

answered Mar 9, 2010 at 8:17

Pascal

21 1

2

Just as reference. Taking from D-B answer and comments, I came with this solution which is close to D-B solution. It works fine in all of my cases and it is thread safe. I don't think that using a ConcurrentDictionary would have been ok.

```
using System;
using System.Collections.Generic;
using System.Xml.Serialization;

namespace HQ.Util.General
{
    public class XmlSerializerHelper
    {
        private static readonly Dictionary<Type, XmlSerializer>
_dictTypeToSerializer = new Dictionary<Type, XmlSerializer>();

        public static XmlSerializer GetSerializer(Type type)
        {
            lock (_dictTypeToSerializer)
            {
                XmlSerializer serializer;
                if (! _dictTypeToSerializer.TryGetValue(type, out serializer))
                {
                    var importer = new XmlReflectionImporter();
                    var mapping = importer.ImportTypeMapping(type, null, null);
                    serializer = new XmlSerializer(mapping);
                    return _dictTypeToSerializer[type] = serializer;
                }

                return serializer;
            }
        }
    }
}
```

Usage:

```
        if (File.Exists(Path))
        {
            using (XmlTextReader reader = new XmlTextReader(Path))
            {
                // XmlSerializer x  = new XmlSerializer(typeof(T));
                var x = XmlSerializerHelper.GetSerializer(typeof(T));

                try
                {
                    options = (OptionsBase<T>)x.Deserialize(reader);
```

```
                }
                catch (Exception ex)
                {
                    Log.Instance.AddEntry(LogType.LogException, "Unable to open
Options file: " + Path, ex);
                }
            }
        }
```

Share Improve this answer Follow edited Oct 24, 2017 at 18:23 answered Oct 24, 2017 at 17:56

Eric Ouellet
11.3k 13 88 129

1

A custom class to serialise:

```
[Serializable]
public class TestClass
{
    int x = 2;
    int y = 4;
    public TestClass(){}
    public TestClass(int x, int y)
    {
        this.x = x;
        this.y = y;
    }

    public int TestFunction()
    {
        return x + y;
    }
}
```

I have attached the code snippet. Maybe this can help you out.

```
static void Main(string[] args)
{
    XmlSerializer xmlSerializer = new XmlSerializer(typeof(TestClass));

    MemoryStream memoryStream = new MemoryStream();
    XmlTextWriter xmlWriter = new XmlTextWriter(memoryStream, Encoding.UTF8);

    TestClass domain = new TestClass(10, 3);
    xmlSerializer.Serialize(xmlWriter, domain);
    memoryStream = (MemoryStream)xmlWriter.BaseStream;
    string xmlSerializedString = ConvertByteArray2Str(memoryStream.ToArray());

    TestClass xmlDomain = (TestClass)DeserializeObject(xmlSerializedString);

    Console.WriteLine(xmlDomain.TestFunction().ToString());
    Console.ReadLine();
}
```

Share Improve this answer Follow

edited Nov 13, 2013 at 19:22

Peter Mortensen
31k 22 108 132

answered Jul 25, 2009 at 15:53

shahjapan
14.1k 22 75 105

2 -1 for no using blocks to prevent resource leaks, and for using XmlTextWriter. – John Saunders Jul 25, 2009 at 16:09

ok agree, but still I have used XmlSerializer xmlSerializer = new XmlSerializer(typeof(TestClass)); but I'm not getting the said Exception. – shahjapan Jul 31, 2009 at 20:15

1

I was having a similar problem, and ignoring the exception did not work for me. My code was calling NServiceBus' configuration `Configure.With(...).XmlSerializer()...`

What fixed it for me was to change the platform for my project.

1. Go to Build\Configuration Manager...
2. Find your project and change Platform (in my case from x86 to Any CPU)

Share Improve this answer Follow

answered Sep 8, 2016 at 12:19

kkelley
11 1

1

Seen a lot of recommendations to use a `ConcurrentDictionary`, but no solid examples of it, so I'm going to throw my hat into this solution race. I'm not a thread-safe developer, so if this code isn't solid, please speak up for the sake of those who follow after.

```
public static class XmlSerializerHelper
{
    private static readonly ConcurrentDictionary<Type, XmlSerializer>
TypeSerializers = new ConcurrentDictionary<Type, XmlSerializer>();

    public static XmlSerializer GetSerializer(Type type)
    {
        return TypeSerializers.GetOrAdd(type,
        t =>
        {
            var importer = new XmlReflectionImporter();
            var mapping = importer.ImportTypeMapping(t, null, null);
            return new XmlSerializer(mapping);
        });
    }
}
```

I've seen other posts involving `ConcurrentDictionary` and `Lazy` loading the value. I'm not sure if that's relevant here or not, but here's the code for that:

```
private static readonly ConcurrentDictionary<Type, Lazy<XmlSerializer>>
TypeSerializers = new ConcurrentDictionary<Type, Lazy<XmlSerializer>>();
```

```
public static XmlSerializer GetSerializer(Type type)
{
    return TypeSerializers.GetOrAdd(type,
    t =>
    {
        var importer = new XmlReflectionImporter();
        var mapping = importer.ImportTypeMapping(t, null, null);
        var lazyResult = new Lazy<XmlSerializer>(() => new
XmlSerializer(mapping), LazyThreadSafetyMode.ExecutionAndPublication);
        return lazyResult;
    }).Value;
}
```

Share Improve this answer Follow edited Aug 29, 2019 at 13:19 answered Aug 29, 2019 at 12:59

Airn5475
2,462 31 51

1

Had a similar problem in one of my .Net Standard dlls.

I used Microsoft.XmlSerializer.Generator nuget, which pre-generating XmlSerializer on .Net Core and .Net Standard.

Share Improve this answer Follow answered Oct 27, 2020 at 13:59

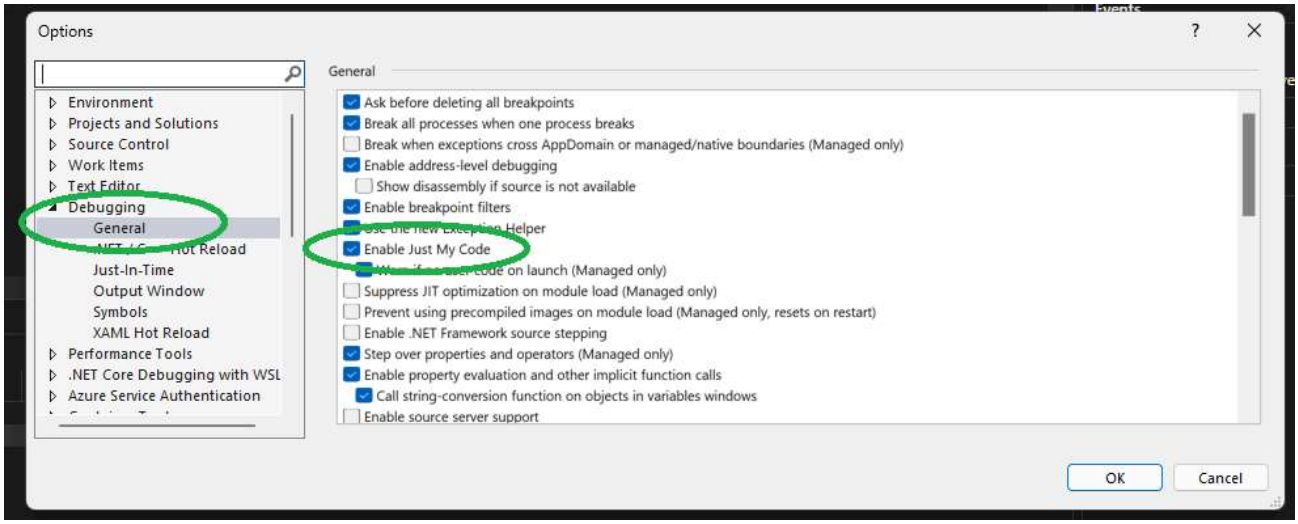
R.G
37 5

Yep, I just had to do that when converting my project to .NET 7. – SolarBear Jan 24 at 22:03

1

> This exception is a part of the XmlSerializer's normal operation. It is expected and will be caught and handled inside of the Framework code.

To see only exceptions that are thrown in your own code (and those that "bubble up" to your code because they are not caught by the library that throws them), use "**Enable Just My Code**":



(The accepted answer ignores the FileNotFoundException, but that is a bad practice.)

Share Improve this answer Follow

answered Apr 21, 2023 at 8:11

SymboLinker
962 7 15

0

Your type may reference other assemblies which cannot be found neither in the GAC nor in your local bin folder ==> ...

> "or one of its dependencies. The system cannot find the file specified"

Can you give an example of the type you want to serialize?

Note: Ensure that your type implements Serializable.

Share Improve this answer Follow

edited Nov 13, 2013 at 19:21

Peter Mortensen
31k 22 108 132

answered Jul 24, 2009 at 9:00

Henrik
568 4 10

0

I was getting the same error, and it was due to the type I was trying to deserialize not having a *default parameterless constructor*. I added a constructor, and it started working.

Share Improve this answer Follow

edited Nov 13, 2013 at 19:28

Peter Mortensen
31k 22 108 132

answered Nov 25, 2011 at 7:33

kay.one
7,642 7 55 75

0

I had the same problem until I used a 3rd Party tool to generate the Class from the XSD and it worked! I discovered that the tool was adding some extra code at the top of my class. When I added this same code to the top of my original class it worked. Here's what I added...

```
#pragma warning disable
namespace MyNamespace
{
  using System;
  using System.Diagnostics;
  using System.Xml.Serialization;
  using System.Collections;
  using System.Xml.Schema;
  using System.ComponentModel;
  using System.Xml;
  using System.Collections.Generic;

  [System.CodeDom.Compiler.GeneratedCodeAttribute("System.Xml", "4.6.1064.2")]
  [System.SerializableAttribute()]
  [System.Diagnostics.DebuggerStepThroughAttribute()]
  [System.ComponentModel.DesignerCategoryAttribute("code")]
  [System.Xml.Serialization.XmlTypeAttribute(AnonymousType = true)]
  [System.Xml.Serialization.XmlRootAttribute(Namespace = "", IsNullable = false)]
  public partial class MyClassName
  {
  ...
```

Share Improve this answer Follow

edited Mar 24, 2016 at 17:40

answered Mar 18, 2016 at 18:29

TheJonz
404 3 11

0

Initial answer from Martin Sheburn is correct. Code samples from edeboursetty, tomas-kubes), quadfinity should solve the problem of not raising excess exceptions in debugger.

Here is a shorter solution, however:

```
internal sealed static class XmlSerializerHelper
{
    private static readonly ConcurrentDictionary<Type,
System.Xml.Serialization.XmlSerializer> s_xmlSerializers = new();

    public static System.Xml.Serialization.XmlSerializer Get<T>()
    {
        return s_xmlSerializers.GetOrAdd(typeof(T), _ =>
System.Xml.Serialization.XmlSerializer.FromTypes(new [] {typeof(T)})[0]);
    }
}
```

Share Improve this answer Follow edited Jan 16, 2023 at 11:14 answered Jan 15, 2023 at 11:35

egors
110 5

0

I had this problem in vb.net when trying to save My.Settings. The setting in question is of type Specialized.StringCollection.

The exception occurred on the call to My.Settings.Save. When I tried to read the settings after restarting the program, the settings were empty (variable showed up in debugger as *Nothing*).

In fact, it seems that the settings *were* being saved, but that they were failing to load. The solution for me was simply to set some initial values in the Settings page of Visual Studio. The settings that I thought had previously failed to be saved then loaded OK (not the initial values I had just entered). The exception still occurs but when retrieving the setting, rather than saving.

Share Improve this answer Follow answered Mar 8, 2023 at 11:14

Colin Fraser
1