Read binary file into a struct

Asked 16 years, 4 months ago Modified 2 years, 9 months ago Viewed 78k times

61

I'm trying to read binary data using C#. I have all the information about the layout of the data in the files I want to read. I'm able to read the data "chunk by chunk", i.e. getting the first 40 bytes of data converting it to a string, get the next 40 bytes.

Since there are at least three slightly different version of the data, I would like to read the data directly into a struct. It just feels so much more right than by reading it "line by line".

I have tried the following approach but to no avail:

```
StructType aStruct;
int count = Marshal.SizeOf(typeof(StructType));
byte[] readBuffer = new byte[count];
BinaryReader reader = new BinaryReader(stream);
readBuffer = reader.ReadBytes(count);
GCHandle handle = GCHandle.Alloc(readBuffer, GCHandleType.Pinned);
aStruct = (StructType) Marshal.PtrToStructure(handle.AddrOfPinnedObject(),
typeof(StructType));
handle.Free();
```

The stream is an opened FileStream from which I have began to read from. I get an `AccessViolationExceptio`n when using `Marshal.PtrToStructure`.

The stream contains more information than I'm trying to read since I'm not interested in data at the end of the file.

The struct is defined like:

```
[StructLayout(LayoutKind.Explicit)]
struct StructType
{
    [FieldOffset(0)]
    public string FileDate;
    [FieldOffset(8)]
    public string FileTime;
    [FieldOffset(16)]
    public int Id1;
    [FieldOffset(20)]
    public string Id2;
}
```

The examples code is changed from original to make this question shorter.

How would I read binary data from a file into a struct?

c# struct io binaryfiles

Share
Improve this question
Follow

edited Dec 21, 2017 at 14:57
EM-Creations
4,291 ● 5 ● 44 ● 58

asked Aug 5, 2008 at 14:28

Robert Höglund
10.1k ● 13 ● 54 ● 70

8 Answers

Sorted by: Highest score (default)

36

+100

The problem is the **string**s in your struct. I found that marshaling types like byte/short/int is not a problem; but when you need to marshal into a complex type such as a string, you need your struct to explicitly mimic an unmanaged type. You can do this with the MarshalAs attrib.

For your example, the following should work:

```
[StructLayout(LayoutKind.Explicit)]
struct StructType
{
    [FieldOffset(0)]
    [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 8)]
    public string FileDate;

    [FieldOffset(8)]
    [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 8)]
    public string FileTime;

    [FieldOffset(16)]
    public int Id1;

    [FieldOffset(20)]
    [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 66)] //Or however long Id2
is.
    public string Id2;
}
```

Share Improve this answer Follow

answered Aug 21, 2008 at 19:02

Ishmaeel
14.4k ● 9 ● 73 ● 84

It looks like it's only possible if the FieldOffset of the string is aligned properly. I had a string at FieldOffset(1) and I couldn't get it to work (resulted in a TypeLoadException).
– Thomas Weller Oct 20, 2022 at 21:37

16

Here is what I am using.
This worked successfully for me for reading Portable Executable Format.
It's a generic function, so `T` is your `struct` type.

```
public static T ByteToType<T>(BinaryReader reader)
{
    byte[] bytes = reader.ReadBytes(Marshal.SizeOf(typeof(T)));

    GCHandle handle = GCHandle.Alloc(bytes, GCHandleType.Pinned);
    T theStructure = (T)Marshal.PtrToStructure(handle.AddrOfPinnedObject(),
typeof(T));
    handle.Free();

    return theStructure;
}
```

Share
Improve this answer
Follow

edited Aug 31, 2015 at 16:08

user3666197
1

answered Nov 2, 2010 at 2:40

Sergey

7

As Ronnie said, I'd use BinaryReader and read each field individually. I can't find the link to the article with this info, but it's been observed that using BinaryReader to read each individual field can be faster than Marshal.PtrToStruct, if the struct contains less than 30-40 or so fields. I'll post the link to the article when I find it.

The article's link is at: http://www.codeproject.com/Articles/10750/Fast-Binary-File-Reading-with-C

When marshaling an array of structs, PtrToStruct gains the upper-hand more quickly, because you can think of the field count as fields * array length.

Share
Improve this answer
Follow

edited Sep 1, 2012 at 17:34

Community Bot
1 ● 1

answered May 6, 2010 at 1:04

nevelis
748 ● 6 ● 17

2 I was just reading: codeproject.com/KB/files/fastbinaryfileinput.aspx. Is this the article you're thinking of? The author notes: "I found that, at about 40 fields, the results for the three approaches were almost equivalent, and beyond that, the block reading approaches gained an upper hand." – Neal Stublen Jun 10, 2010 at 19:39

3

I don't see any problem with your code.

just out of my head, what if you try to do it manually? does it work?

```
BinaryReader reader = new BinaryReader(stream);
StructType o = new StructType();
o.FileDate = Encoding.ASCII.GetString(reader.ReadBytes(8));
o.FileTime = Encoding.ASCII.GetString(reader.ReadBytes(8));
...
...
...
```

also try

```
StructType o = new StructType();
byte[] buffer = new byte[Marshal.SizeOf(typeof(StructType))];
GCHandle handle = GCHandle.Alloc(buffer, GCHandleType.Pinned);
Marshal.StructureToPtr(o, handle.AddrOfPinnedObject(), false);
handle.Free();
```

then use **buffer[]** in your BinaryReader instead of reading data from FileStream to see whether you still get AccessViolation exception.

> I had no luck using the BinaryFormatter, I guess I have to have a complete struct that matches the content of the file exactly.

That makes sense, BinaryFormatter has its own data format, completely incompatible with yours.

Share

Improve this answer

Follow

edited Aug 6, 2008 at 9:13

answered Aug 5, 2008 at 15:31

lubos hasko

25k ● 10 ● 57 ● 62

3

I had no luck using the BinaryFormatter, I guess I have to have a complete struct that matches the content of the file exactly. I realised that in the end I wasn't interested in very much of the file content anyway so I went with the solution of reading part of stream into a bytebuffer and then converting it using

```
Encoding.ASCII.GetString()
```

for strings and

```
BitConverter.ToInt32()
```

for the integers.

I will need to be able to parse more of the file later on but for this version I got away with just a couple of lines of code.

Share
Improve this answer
Follow

edited Jan 7, 2012 at 0:53

thkala
86.2k • 24 • 164 • 204

answered Aug 6, 2008 at 9:03

Robert Höglund
10.1k • 13 • 54 • 70

0

Try this:

```
using (FileStream stream = new FileStream(fileName, FileMode.Open))
{
    BinaryFormatter formatter = new BinaryFormatter();
    StructType aStruct = (StructType)formatter.Deserialize(filestream);
}
```

Share Improve this answer Follow

answered Aug 5, 2008 at 14:56

urini
33k • 14 • 42 • 37

7 BinaryFormatter has its own format for binary data - which is fine if you are reading/writing the data yourself. not useful if you are getting a file from another source. – russau Jul 26, 2009 at 7:11

0

Reading straight into structs is evil - many a C program has fallen over because of different byte orderings, different compiler implementations of fields, packing, word size.....

You are best of serialising and deserialising byte by byte. Use the build in stuff if you want or just get used to BinaryReader.

Share Improve this answer Follow

answered Sep 23, 2008 at 21:43

Ronnie
8,107 • 6 • 35 • 34

6 I disagree, reading straight into structs is sometimes the fastest way to get your data into a usable object. If you're writing performance oriented code this can be very useful. Yes you must be aware of alignments and packing and be sure any endpoint machine will use the same. – Joe Feb 3, 2012 at 20:00

3 I also disagree. When performance is key, or when you need binary C++/C# interop, writing plain `struct`s is the way to go. – Dmitri Nesteruk Mar 25, 2012 at 7:48

"Use the built in stuff" would be a helpful answer if the built in stuff was explained. C# seems to lack anything like easy to use built in stuff. Extra true if what you are reading in is broken

down into bitfields. – JamieB Feb 20, 2023 at 15:29

0

I had structure:

```
[StructLayout(LayoutKind.Explicit, Size = 21)]
    public struct RecordStruct
    {
        [FieldOffset(0)]
        public double Var1;

        [FieldOffset(8)]
        public byte var2

        [FieldOffset(9)]
        [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 12)]
        public string String1;
    }
}
```

and I received **"incorrectly aligned or overlapped by non-object"**. Based on that I found: https://social.msdn.microsoft.com/Forums/vstudio/en-US/2f9ffce5-4c64-4ea7-a994-06b372b28c39/strange-issue-with-layoutkindexplicit?forum=clr

> OK. I think I understand what's going on here. It seems like the problem is related to the fact that the array type (which is an object type) must be stored at a 4-byte boundary in memory. However, what you're really trying to do is serialize the 6 bytes separately.
>
> I think the problem is the mix between FieldOffset and serialization rules. I'm thinking that structlayout.sequential may work for you, since it doesn't actually modify the in-memory representation of the structure. I think FieldOffset is actually modifying the in-memory layout of the type. This causes problems because the .NET framework requires object references to be aligned on appropriate boundaries (it seems).

So my struct was defined as explicit with:

```
[StructLayout(LayoutKind.Explicit, Size = 21)]
```

and thus my fields had specified

```
[FieldOffset(<offset_number>)]
```

but when you change your struct to Sequentional, you can get rid of those offsets and the error will disappear. Something like:

```
[StructLayout(LayoutKind.Sequential, Size = 21)]
    public struct RecordStruct
    {
        public double Var1;

        public byte var2;

        [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 12)]
        public string String1;
    }
}
```

Share Improve this answer Follow

answered Mar 13, 2022 at 12:38

Kebechet
2,296 ● 2 ● 26 ● 41