

# Re: reading a C++ structure from a binary file using C#

---

*Source:*

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.languages.csharp/2007-03/msg04906.html>

---

- *From:* "Willy Denoyette [MVP]" <[willy.denoyette@xxxxxxxxxx](mailto:willy.denoyette@xxxxxxxxxx)>
  - *Date:* Fri, 30 Mar 2007 22:03:59 +0200
- 

"Ben Voigt" <[rbv@xxxxxxxxxxxxxx](mailto:rbv@xxxxxxxxxxxxxx)> wrote in message  
[news:ee\\$9PDwcHHA.1000@xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx](mailto:news:ee$9PDwcHHA.1000@xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx)

"Willy Denoyette [MVP]" <[willy.denoyette@xxxxxxxxxx](mailto:willy.denoyette@xxxxxxxxxx)> wrote in message  
[news:eDerzTucHHA.4632@xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx](mailto:news:eDerzTucHHA.4632@xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx)

"Ben Voigt" <[rbv@xxxxxxxxxxxxxx](mailto:rbv@xxxxxxxxxxxxxx)> wrote in message  
[news:eUJz4%23tcHHA.208@xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx](mailto:news:eUJz4%23tcHHA.208@xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx)

"Nicholas Paldino [.NET/C# MVP]"  
<[mvp@xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx](mailto:mvp@xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx)> wrote in message  
[news:uklz82tcHHA.4260@xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx](mailto:news:uklz82tcHHA.4260@xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx)

Nand,

In order to do this in .NET, I would read from the file in 74 byte blocks (4 + 20 + 50). Then, I would call the static ToInt32 method on the

The C++ code used sizeof(s)... you must print this value out from C++. It is unlikely to be 74, since the structure contains an int member which prefers 4-byte alignment... so I think two bytes of padding is added by the C++ compiler and the record spacing will be 76 bytes. But check it, because it pragma pack was used several different values would be possible.

Why? The int is the first field of the struct, the second is a char array which has no alignment restriction, so, whatever the packing, the length written on disk will be 74.

## Re: reading a C++ structure from a binary file using C#

As I said, the OP's C code writes sizeof(s) bytes to disk. Padding is included in sizeof, because sizeof is the separation between adjacent array elements which must both be properly aligned.

If there's only one record in the file, it's a non-issue. But I was responding to a post mentioning "read the file in 74 byte blocks".

Oh, I see what you mean, but here the sizeof depends on the packing specified (or the default packing). This is why I'm always trying to define my structs like this;

```
#pragma pack(show)
struct student
{
int roll_no;
char name[20];
char qualification[50];
};
#pragma pack(pop)
#pragma pack(show)
```

when writing to disk, especially when these files have to be read by "foreign" applications. No surprises here, the size is exactly the sum of all it's elements.

Willy.

.