

Re: BinaryWriter (or streams) slow performance

Source:

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.framework.performance/2007-02/msg00042.htm>

- *From:* Jon Skeet [C# MVP] <skeet@xxxxxxxx>
 - *Date:* Tue, 27 Feb 2007 09:31:42 -0000
-

Rock2000 <Rock2000@xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx> wrote:

1) Getting the Position of the baseStream of the writer takes more then no-time. Why isn't this cached?

Who says you're the only one writing to that stream?

I'm not sure I know what you mean. The stream has the Position right. So even if 3 clients were using it, the shared stream should still know what the current position is. Plus I opened it with no sharing if that helps.

Ah, I see what you mean, yes. I suspect there are still some complexities depending on the kind of underlying stream – my guess is that for file streams, it goes straight down to Win32 anyway.

2) Seeking to the same location that I'm already at takes longer then zero time. Granted I should try to avoid that, but shouldn't this be a no-op essentially? Maybe the problem is really back to 1.

Depends on how you're doing it, but yes, see 1. Sounds like something that's worth trying to avoid anyway though.

Yeah I won't disagree that I shouldn't be doing that, but sometimes with code hierarchies and encapsulation, the context isn't there to know these things. In my case I could get around this. Just seemed odd.

Fair enough.

Re: BinaryWriter (or streams) slow performance

3) SEEMS like the writer is flushing any buffered data during a seek.
Shouldn't it try to update the buffer if it hasn't been written out yet and
the seek isn't seeking out of the buffered area?

How are you trying to seek? But no, I believe that flushing is a perfectly valid behaviour.

I'm basically converting some Java code to C#, and for this particular module I'm creating a file with a certain format from multiple input files. The format is certainly not pretty, but essentially the code is writing a start block, which contains a dummy int value for the block size, then it's writing the block data, and then seeking back up to fill in the block size which it now knows. Then seeking back to where it was to continue (yeah :(, I didn't design it). It does this for a ton of nested blocks. The problem is it doesn't know the size of the block before writing it, and I don't want to go and redesign it.

Are the blocks small enough that you could create the block *and header* in memory, and then write out the whole block in one go?

Most of the seeks are not far away (the blocks are generally pretty small), so if the writer saw that it had the previous 'dummy' value still in the writer buffer, it could seek back to it, modify it in memory, and there's almost no overhead. When it gets flushed it will be correct. That's what I did manually anyways. Just flushing on a seek isn't 'wrong' certainly, but I assumed something this core to the framework would be a little more sophisticated and optimized. Currently it seems pretty simplistic (but certainly there are many aspects of IO I'm sure I don't use that may preclude these types of optimizations)

That's the thing – while I couldn't point to any subtleties right now, I bet there are plenty lurking around...

—
Jon Skeet – <skeet@xxxxxxxx>
<http://www.pobox.com/~skeet> Blog: http://www.ms_mvps.com/jon.skeet
If replying to the group, please do not mail me too