

Re: Dispose(bool), IDisposable, form closing etc.

Re: Dispose(bool), IDisposable, form closing etc.

Source:

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.languages.csharp/2007-11/msg03482.html>

- *From:* Brian Gideon <briangideon@xxxxxxxx>
 - *Date:* Mon, 26 Nov 2007 06:46:32 -0800 (PST)
-

On Nov 21, 12:26 pm, Peter Duniho <NpOeStPe...@xxxxxxxxxxxxxxxxxxxx> wrote:

It is possible that the CLI `_allows_` for early collection of parameter-held references, and that the current implementation itself just doesn't take advantage of that. But if so, the behavior is (I think) more important than the theoretical possibility in this case, since we're talking about what `Application.Run()` `_does_`, not what it might do.

Maybe. I'm just thinking about theoretical possibilities that I could not demonstrate in 1.1, but were a piece of cake in 2.0. At some level you have to consider the possibility or risk hard to find bugs when your code runs on a newer version of the framework.

Obviously, a parameter can be collected during the execution of an unmanaged function.

Based on what I've seen, I'm not convinced of that. Even when there's no actual use of the parameter in a method, the object referenced by it is apparently not collectable until the method returns. It wouldn't matter whether you called a managed or unmanaged function.

Yeah, after some more thought I think you're right. The interop marshaler would likely prevent the reference from being eligible until the function returns. Asynchronous functions and functions that accept say...an `IntPtr` to a managed object would still be problematic.

That's the most important reason for the existence of `GC.KeepAlive`.

Re: Dispose(bool), IDisposable, form closing etc.

Re: Dispose(bool), IDisposable, form closing etc.

I'm not sure I'd rank the reasons in that way, but I'm not willing to quibble about which is the `_most_` important. Clearly you need `GC.KeepAlive()` any time you need to avoid early collection of references that aren't explicitly used later in the method. There are lots of different ways to run into the issue, and calling an unmanaged function is just one example.

But if you want that to be the most important, that's okay by me. :)

No, you are right. I should be careful about ranking things like that. There certainly are other reasons for it's use. A scenario involving an unmanaged function just happened to be the only time I've used it :)

.