

Re: Problem with C# not calling MC++ destrcutor

Source:

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.languages.csharp/2007-02/msg00348.html>

- *From:* Göran Andersson <guffa@xxxxxxxxxx>
 - *Date:* Fri, 02 Feb 2007 21:54:37 +0100
-

Jeremy Chaney wrote:

Lebesgue wrote:

I'm pretty surprised to hear that it is possible for my Finalizers to not be called at shutdown. If I am being required to manually free up my resources, what is the point of GC?

GC is here to clean up unreferenced `_memory_`. You should deal with other (unmanaged) resources using the dispose pattern in C#. There has been a long thread recently (named C++/CLI is the way to go) where a reference counted model for disposing unused resources in C# has been proposed.

Can you show us your finalizers? My guess is you are trying to use them the way they were not meant to be.

I think I'm sinking in over my head here... :) When you say Finalizers, do you mean my "~MyClass" method?

Yes, he does.

Here it is:

```
~MyCSharpClass()
{
    foreach (KeyValuePair<Char, MyManagedCPPClass> kvp in m_MyMap)
    {
        kvp.Value.Dispose();
    }
    m_myotherManagedCPPClass.Dispose();
}
```

Re: Problem with C# not calling MC++ destrcutor

The need for this destructor (I'm going to call it that until someone tells me what to call it in C#) is surprising to me. The objects that I have to call Dispose on are both Managed C++ objects.

As you use objects that implement IDisposable, your class should also implement IDisposable, so that the resurces can be freed safely and as soon as possible.

If you are using the finalizer to free the resources, you have absolutely no control over when it happens. If you implement IDisposable, you can call the Dispose method when you want to free the resources.

Shouldn't they
have their destructors called when "~MyCSharpClass" is called?

No. After your finalizer has run, there will be no references to the child objects. At the next garbage collection they will be placed in the queue of object to be finalized, and a background process will call their finalizers eventually.

If you implement IDisposable (with Dispose containing the code that you currently have in the finalizer), it will make the memory management much more efficient. After you have called Dispose, there are no finalizers to handle, and the garbage collector can free the memory of your object and all it's child objects at the same time.

Before
I added the calls to Dispose, "~MyCSharpClass" was always called, but the destructors for my Managed C++ objects were not. (I fixed this with the addition of Dispose).

Thanks,
--Jeremy

--
Göran Andersson

<http://www.guffa.com>
.