

# Re: Clean Up managed resources

---

*Source:*

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.languages.csharp/2007-05/msg03463.html>

---

- *From:* "Ben Voigt" <rbv@xxxxxxxxxxxxxx>
  - *Date:* Wed, 23 May 2007 08:13:16 -0500
- 

"Koliber (js)" <profesor\_kinbote@xxxxxxxxxxxxxx> wrote in message  
<news:1179916880.043144.32910@xx>

On 23 Maj, 01:49, "Ben Voigt" <r...@xxxxxxxxxxxxxx> wrote:

```
Public class MyClass:IDisposable
{
private bool IsDisposed=false;
public void Dispose()
{
Dispose(true);
GC.SuppressFinalize(this);
}
```

```
protected void Dispose(bool Diping)
{
if(!IsDisposed)
{
if(Diping)
{
//Clean Up managed resources
////////////////////////////////////
```

```
aMember.Dispose();
bMember.Dispose();
cMember.Dispose();
```

```
////////////////////////////////////
}
//Clean up unmanaged resources
////////////////////////////////////
```

## Re: Clean Up managed resources

some pinvoke resource releases  
for example 'paired' api calls  
like this from this page

<http://safari.oreilly.com/0321174038/app04>

```
////////////////////////////////////  
}  
IsDisposed=true;  
}
```

```
~MyClass()  
{  
Dispose(false);  
}
```

Is that right?  
-----

Yes, that is it.

The difference lies not in whether there is an unmanaged resource. Of course there is an unmanaged resource. SqlConnection, File, Socket, all of these have unmanaged resources. The point is to get them cleaned up at the right time without messing up the garbage collector.

If somebody called Dispose on your object, then it is still reachable. Therefore you should ask all the resources to dispose as well.

If the finalizer is called on your object, then it is not reachable. If you have private resources, then they are also not reachable. If they are .NET classes, then they have their own finalizers which will be run automatically at about the same time as yours, you do not need to call them. In fact it would be an error to do so because:

## Re: Clean Up managed resources

- (1) their finalizers might already have run
- (2) if you access the object, you will prevent it from being freed during this garbage collection cycle (resurrection)
- (3) if you are subscribed to any events of the object, and resurrect it, then it will resurrect your object as well

The key thing is, your client only tells you to dispose, so you have to pass the message along. The garbage collector will tell each managed class to finalize, so you don't tell other .NET classes when you are being finalized.

But the garbage collector doesn't know about anything except managed classes, so if you called `p/invoke`, you need to clean that up yourself.

Ok?

Ok, I understand this I think, but I have maybe a little hesittion to this :>

If my object finalizes oneself and thus my dispose not calls my members disposes – who calls them? It is only hesitation because I suppose that if any object in this chain has a finaliser who do clean up for self – not for its members – it also shoud be ok. Is that right?

The other thing I do not understand and I want to ask maybe is: what is a mechanism of this famous theme: "if you do a finaliser in your class there memory clean up will work slower because gc must then block something and waits for something" Can someone explain it to me maybe?

I haven't tried to measure, but I don't think having a finalizer would really be that much of a problem. The GC already has to block all other threads while it determines reachability from all the different roots.

The problem is that, if your finalizer does anything at all, it will need access to its member variables. Then a tracking handle to your object needs to be placed on the stack and now your object is reachable again. So the GC can't free your memory. After all, your finalizer could do `some_global_list.Add(this)!`

thanx in advance  
K.

Re: Clean Up managed resources