

Re: Threading problem with Garbage Collector

Source:

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.languages.csharp/2006-01/msg03161.html>

- *From:* "Christian Kaiser" <chk@xxxxxxxxxx>
 - *Date:* Tue, 17 Jan 2006 23:10:17 +0100
-

Hi Willy,

OK, sorry, back to square one, I'll try to be more specific ;-)

The component is a managed C# control. It itself controls calls to a DLL (written in Delphi, but that itself is not the basic problem) via PInvoke. So the C# class is meant to be a proxy to the DLL's functions, some more or less slim wrapper around its functions.

When the component is created (in the main thread, let's call it thread "A"), it loads the DLL, which in turn then creates a window (it needs to, unfortunately). It does so right after LoadLibrary of the DLL, in the startup code of the DLL (while initializing static variables, ...).

The problem comes when the component is disposed of by the Garbage Collector, which has a different thread "B". The Dispose() of the component unloads the DLL, which tries to destroy its window. And this now is the problem – the DLL called by thread "B" cannot destroy the window created in thread "A". Here it's the problem of Delphi, I assume, but the problem is more or less valid in other languages too. DestroyWindow() does not work, sendin a WM_CLOSE still lets the window survive, ... -> GPF.

The solution we thought about then was to create a window in our component (using a "Label" for example) to be able to switch from thread "B" to thread "A" in a standard way – synchronous access to a control's functions. So in its constructor, the component wants to create the label window, and in the Dispose() method it uses the label window to synchronize the FreeLibrary() by invoking a delegate, which calls an event of the label, which in turn (as it's in thread A by then) can free the DLL as it should be. Draft (from memory, I'm at home now):

```
delegate void DelegateKillDLL();
```

```
class component
{
void KillDLL()
{
```

Re: Threading problem with Garbage Collector

```
// Free external DLL...
}

component()
{
myLabel = new Label();
// Load external DLL.
}

void Dispose()
{
DelegateKillDLL dlg = new DelegateKillDLL(KillDLL);
myLabel.Invoke(dlg, object[] {});
}
}
```

a) the Invoke() call does not work, as "new Label()" did not create a window. So "Invoke()" throws an exception that it cannot work without any window. Is there a way to create a window for such a purpose (WS_OVERLAPPED style, so to speak)? Our component is not a "visual" control (it does not have any window, is not derived from a visual component, the "Container" property is null).

b) will the whole idea work in a GC thread?

c) Is there a better way to do this?

I hope I was able to describe it better than before. If there's any question, don't hesitate to ask. I'm relatively new to C# (but very experienced in C++, Win32, ...)

Christian

```
"Willy Denoyette [MVP]" <willy.denoyette@xxxxxxxxxx> wrote in message
news:OE4Xfd4GGHA.3036@xxxxxxxxxxxxxxxxxxxxxxxxxxxx
>
> "Christian Kaiser" <bchk@xxxxxx> wrote in message
> news:eTnuoE4GGHA.2040@xxxxxxxxxxxxxxxxxxxxxxxxxxxx
> | We have a component that has no window. Well, no window in managed
> code –
> it
> | uses a DLL which itself uses a window, and this is our problem!
> |
>
> Components and DLL's are such general terms that say so little about what
> they actually are, so you will have to be more explicit.
> What's exactly the component? I guess it's a managed class.
> What's contained in the DLL, how do you access the methods/functions?
> Through PInvoke or COM interop?
```

Re: Threading problem with Garbage Collector

Re: Threading problem with Garbage Collector

> If COM
> – what are its threading requirements (apartment type)?
> – What is the apartment state of the thread on which it is created?
>
> | When the garbage collector runs and removes our component (created
> | dynamically by, say, a button click, and then not referenced any more),
> | the
> | GC runs in a different thread, which prohibits the DLL to destroy its
> | window, resulting in a GPF when the WndProc of that window is called –
> | the
> | code is gone (DLL unloaded), but the window still there....
> |
>
> Hmm... who's unloading the DLL? The GC doesn't unload DLL's.
>
>
> | Clear to see (took us hours, to be honest, to understand what caused the
> | GPF, as WinDBG somehow uses 100% CPU load if used with .NET 2.0), which
> | was
> | a PITA and a lot of guesswork.
> |
> | Is there a way to get around that problem, so that we can call a routine
> | from Dispose() in the context of the original thread?
> |
> | We tried creating a window ("m_myLabel = new Label()") in the
> | component's
> | constructor, and use a delegate and "m_myLabel.Invoke(...)" this to get
> | a
> | synchronous invoke in the original thread. Problem is, the component
> | has
> | no
> | form to create the "Label" window with, thus invoke throws an exception
> | telling us that a non-existing window does not support Invoke. Right it
> | is.
> |
> | Questions:
> |
> | a) How can a window be created without a parent form (invisible,
> | independent, just for this delegate), or
> |
>
> You can't, and it's not needed, your DLL code creates the window so it's
> responsible for its life time.
>
> | b) is there a better way to tell the GC to use the original "main"
> | thread?
> |
>
> No, and this is not the problem anyway, the GC doesn't know about
> unmanaged
> stuff.

Re: Threading problem with Garbage Collector

> I guess your problem relates to threading issues with COM interop, but I
> could be wrong of course.
>
> Willy.
>
>

- **Follow-Ups:**

- ◆ **Re: Threading problem with Garbage Collector**
◇ From: Willy Denoyette [MVP]

- **References:**

- ◆ **Threading problem with Garbage Collector**
◇ From: Christian Kaiser
- ◆ **Re: Threading problem with Garbage Collector**
◇ From: Willy Denoyette [MVP]

- Prev by Date: **Set Anchor on Mdi child**
- Next by Date: **Re: Set Anchor on Mdi child**
- Previous by thread: **Re: Threading problem with Garbage Collector**
- Next by thread: **Re: Threading problem with Garbage Collector**
- Index(es):
 - ◆ **Date**
 - ◆ **Thread**