

Re: Multithreaded GUI issues

Source:

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.languages.csharp/2007-08/msg02803.html>

- *From:* Peter Duniho <NpOeStPeAdM@xxxxxxxxxxxxxxxxxxxxx>
 - *Date:* Mon, 20 Aug 2007 20:06:09 -0700
-

pedrito wrote:

I posted an issue a couple of days ago that nobody has answered anything on yet (re: error in System.Net.OSSOCK.closesocket()) and I'm starting to wonder if maybe my problem lies elsewhere.

I saw that. I didn't see a post constructed clearly enough to make it worth trying to reply. I can't speak for anyone else, but that's why I didn't bother to comment on it.

I will point out that there wasn't any indication in your post that you were actually trying to close a socket at the time the exception occurred, so you might want to check to see why something is trying to close a socket when you don't expect to.

As I mentioned in that post, the app is multithreaded. It downloads from a number of threads simultaneously. There are 4 or 5 events that can come off of these separate threads.

In each of these cases, where the events get passed on to GUI components, they're Invoked into the GUI thread.

But then it occurred to me, is there a specific "line in the sand"? What GUI methods are callable from separate threads and which ones aren't?

None can. You must use Invoke() or BeginInvoke() on any code that will eventually access a Control-derived instance.

Okay, that's not strictly true. But you should follow that rule, because AFAIK Microsoft hasn't published any guidance regarding what's safe to do and what's not. And also AFAIK practically everything that you might do with an existing .NET Control-derived class needs to be invoked on that Control-derived instance's owning thread (most of the methods and properties in those classes translate to some underlying window message, and that's where the thread-specific requirement comes from).

Obviously if you subclass a Control class, then your own implementation is known to you and you can easily tell what's safe to execute on another thread or not. But for consistency you may want to follow the same rule anyway. The other benefit of doing so is that using Invoke() or BeginInvoke() ensures synchronization of code executing that deals with the Control-derived instance.

Re: Multithreaded GUI issues

For example, some of these threads might download images and in those cases, I'm creating Image objects (on non-gui threads) to get information about the images (dimensions mainly), though they're never drawn. But does this constitute the wrong side of the line?

It depends on what you do with the Image objects and any resulting data (like the dimensions).

Clearly some GUI-side methods can, and should be called from separate threads. `Control.InvokeRequired` and `Control.Invoke()`, obviously.

Anything that is literally affecting the GUI needs to use `Invoke()`.

Is there a list somewhere of what's safe and what's not safe to call from a non-gui thread? I mean, sometimes it's hard to know for sure if something in the framework is eventually going to do GUI side work even though it might not be apparent from the call.

I'm not not really clear on what you mean here. It seems to me that if you assume that anything that uses an instance of a Control-derived class needs to be done on the GUI thread, that is the safest, most reliable practice. Likewise, if your code does not actually access the Control-derived instance, there should be no problem.

Of course, you need to keep track of things you may have done in your code that would indirectly cause access to a Control-derived class. But that would all be in your own implementation. I'm not aware of things that you might do with non-GUI objects that could still affect or otherwise interact with GUI objects.

Whether any of this has anything to do with your original problem, I can't say.

Pete

.