

Re: Want form to show changing data. But it could be closed, or closing, during update.

Re: Want form to show changing data. But it could be closed, or closing, during update.

Source:

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.languages.csharp/2007-05/msg04239.html>

- *From:* "Peter Duniho" <NpOeStPeAdM@xxxxxxxxxxxxxxxxxxxx>
 - *Date:* Mon, 28 May 2007 11:32:21 -0700
-

On Mon, 28 May 2007 10:02:05 -0700, Zytan <zytanlithium@xxxxxxxx> wrote:

You call Invoke() from a static method the same way you'd call from an instance method, except of course you need to explicitly specify the instance on which you want to call Invoke(). Obviously this means you need a reference to the form somewhere.....[snip]

Yup, and that's the issue, to have that reference, and to use it when it could change on you at any moment.

Well, as I mentioned, subscribing to an event is a very common way to implicitly allow some component publishing that event to reference the subscriber (your form in this case). The form instance itself will not disappear as long as it's still subscribed to the event, though you may of course still need some logic to check on whether the form's been disposed. That said, there are probably better ways around the issue.

Pete, the data is coming through a socket connection, and who knows when the data will arrive.

Is it correct to assume that the stream of data exists whether or not the form is open?

Since the form is just to show the current stream of data, and is not the main form, I don't want the form to exist for all time, and be processing all the time (it takes up memory and CPU resources). I want it to allow it to be opened/closed as desired. When open, it will show the current data (but also update in real-time).

A couple of points:

1) I think it's not really all that significant a consumption of memory and CPU resources to keep the form

Re: Want form to show changing data. But it could be closed, or closing, during update.

Re: Want form to show changing data. But it could be closed, or closing, during update.

around all the time. In fact, IMHO this may be your very best solution as its nice and simple and avoids a lot of the synchronization hassles you're dealing with. This definitely falls into the category of "don't waste time optimizing something until you know it's a bottleneck". I doubt that you'd find your form is a bottleneck, especially since it seems that it's okay to have it around at least some of the time.

2) If you insist that you need to be able to open and close the form at will, it may be better to create a producer/consumer type interface, in which a third data structure that is always present maintains the list of data to be displayed. Your i/o code would produce data onto the list, and the form would consume data from the list. The list could include some semantics that the form could apply that cause it to discard data and stop actually enqueueing new data when the form closes. That way, the data structure is always around, but when the form isn't using it, it basically does nothing (just ignoring any input from the i/o code).

Right now, this socket thread checks to see if the form exists, and if so, invokes a method on it to update itself. There's still that little chance the form could disappear right AFTER the existence check, but BEFORE the reference is used to invoke. Things are properly synch'ed AFTER the invoke, but not before, so that's the danger area. (I could catch exceptions to prevent a crash, I suppose).

I think there are better ways around the issue, but at the very least, you could use some sort of synchronization object that is acquired by threads wanting to Invoke() as well as the form's thread when it wants to close itself. The threads would use some sort of "try" semantics on the synchronization object (the exact syntax would depend on the object) so that they can detect the closing case.

But, again...if you fix things so that the form is guaranteed to not close itself until the threads are done invoking methods on it, then the synchronization issue goes away (or rather, it's solved elsewhere, in an easier-to-deal-with way).

Pete

.