

Re: Cross thread question

Source:

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.languages.csharp/2007-02/msg00151.html>

- *From:* "Bruce Wood" <brucewood@xxxxxxxxxxx>
 - *Date:* 1 Feb 2007 12:23:51 -0800
-

On Feb 1, 11:21 am, t...@xxxxxxxxxxx wrote:

Hello,

I have an app that reads data params from a stream and updates controls accordingly. The stream reader is on a different thread than the main thread that created the controls. I fully realize it's not wise, and in some circumstances, not even possible to update the controls from the stream other than the main stream. I believe there are two recommended ways to handle this situation.

1. Use Invoke() to "call" an updater method on the main thread from the stream reader thread.
 2. Use the BackgroundWorker class and update controls with the RunWorkerCompleted event handler.
- Question is: Which one is better? I have to update about 75 controls at a 10Hz rate (10 times a second) so I'm looking for speed here.
TIA

using 2.0, VS2005

Here's another option.

If you know that you're going to be updating the controls rapidly, 10 times a second, as you said, and you're going to do this until, say, the user presses Cancel or something, you could create an intermediate area where the values for the controls are stored. So, conceptually, you have your UI controls, which are displaying results, your stream, which is reading results to display, and a "neutral zone" where the results are stored.

Your stream thread just reads results and updates the "neutral zone" values.

Your UI thread runs in a loop and (at some regular interval) picks up values from the "neutral zone" and updates all of the controls.

Good points:

No need to marshal calls across threads. The two threads run independently. I anticipate that this will be a big time saver.

The UI thread can be tuned (UI updates) separately from the stream thread. In other words, you've decoupled the need for every-input-must-be-displayed logic.

So... the UI can never "get behind" the updates and have a huge number of calls build up on its event queue because there are no calls from the stream thread to the UI thread.

Bad points:

Because input and display are decoupled, any given display may show inconsistent data. However, if you're really updating 10 times a second, I doubt that the user will notice.

The "neutral zone" data area must have proper locking done on it, so that the UI thread doesn't read a value as its being updated by the stream thread.

You may miss displaying some results, if the stream thread updates a result twice before the UI thread displays it. Again, I doubt that your user will notice.

You may do many more UI updates than are necessary. If one UI control changes frequently while another changes rarely, and if this is unpredictable, you will be updating all 75 controls every cycle, because you have no idea what has changed and what hasn't. (This can be mitigated by taking a snapshot of the "neutral zone" and then updating the UI from that, giving you a basis for comparison next time, but that's getting rather complicated, IMHO.)

Anyway... some food for thought.

.