

## Re: Closing out threads

---

*Source:*

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.languages.csharp/2007-09/msg00020.html>

---

- *From:* Lilith <lilith@xxxxxxxx>
  - *Date:* Fri, 31 Aug 2007 23:41:29 -0500
- 

On Fri, 31 Aug 2007 23:24:06 -0500, "pedrito" <pixbypedrito at yahoo.com> wrote:

"Lilith" <lilith@xxxxxxxx> wrote in message  
<news:dqhhd31ldkqke6asuuta8deldgb85n8f1g@xxxxxxxx>

On Fri, 31 Aug 2007 20:13:07 -0500, "pedrito" <pixbypedrito at yahoo.com> wrote:

"Lilith" <lilith@xxxxxxxx> wrote in message  
<news:2qbhd3lelr1uj1mqvjnemoc6shoev1mql@xxxxxxxx>

Maybe I'm making some assumptions about how threads work. I have a form that, on the press of a button, starts monitoring some SMTP connections. On occasions I make changes while the thread is running in the background. To reload the changed options I have to exit the function the thread is running and restart it. I have an indicator that tells me the function is exiting. But on restart I get an exception that, to the best of my understanding, says that the thread is still running and cannot be restarted.

What is the exception?

ThreadStateException. Thread is running or terminated; it cannot restart.

## Re: Closing out threads

Okay, I see, so you are trying to reuse a thread that's completed (terminated). See below..

So, does the thread keep running after I exit the function? Or is there something I should be doing to tell the thread to terminate? Do I need to design the function to run the thread until program's end but read flags from the UI to stop and reload options?

You can't restart a thread after it has exited. You have to start a new thread. If you want to keep the same thread running, then what you ought to do is have some sort of queue that it can access (using some sort of locking mechanism like `lock()` or `Monitor.Enter()/Monitor.Exit()`) and simply have it periodically check to see if there's information it needs, but don't have it exit until the app is done.

Please verify this for me then so I know I understand. A given

```
Thread myThread;
```

can only be initialized/started once. Even when the function specified by `StartThread` is complete, the `myThread` reference cannot be used again?

Exactly. You simply create a new thread and start it up. Just be sure to eliminate any references to the previous thread so the garbage collector can clean it up.

## Re: Closing out threads

The preferred method is to simply keep the thread running and have it pick up updates from some data source. I generally use a queue or an array. It's just important to synchronize access with lock() or Monitor.Enter()/Monitor.Exit() so that two threads don't try to access the same data at the same time.

There are a number of possible issues with regards to threading and you might want to search around and read some of the information out there on writing threaded apps in .NET. I know Jon Skeet (one of the more prolific contributors to this newsgroup) has some terrific information here:  
<http://www.yoda.arachsys.com/csharp/threads/>

In fact, I highly recommend his other writings on various topics here:

<http://www.yoda.arachsys.com/csharp/>

Looks to be a set of good reference/instructional material. This is going into my bookmarks

He's got about the best .NET "kung-fu" around, so the information is very high quality.

Alternatively, you can simply let the thread close and fire off a new thread in its place..

New thread? Using the same reference? I thought that's what I was doing. But apparently my perception of what is going on is cloudy. Or are we talking about a different thread reference?

If you fire off a completely new thread [Thread myThread = new Thread(new ThreadStart(MyThreadMethod))] or however you were firing off your threads, is fine as well. The key there is keeping track of how many concurrent threads you have going. It can be problematic to have too many going at once. There's pooling stuff in .NET (though I personally don't use it much as I find it easier to use less robust pooling/management methods which satisfy my needs in most cases).

Lilith, the ever confounded

P.S. Multitudes of thanks.

Re: Closing out threads

Lilith

Nothing wrong with being confounded. Multithreading opens up a pretty big can of worms and it takes a while to get the hang of it, but I find it to be very worthwhile for a lot of things. In fact, .NET makes threading about as easy as it can really be. Back in my C++ days, threading was something I generally avoided because it was relatively tedious and difficult to manage, but I find myself frequently finding opportunities to use multithreading in .NET and never hesitate to use it because it is relatively easy.

There have been a number of things that I've hesitated to look into because of, I guess, fear of failure. Dynamic memory allocation used to scare me scitless. :-) Once I started using it and learned the foibles it became easier. Threading in C++/MFC was relatively easy to believe I understood it, but I've always had problems with completely understanding the equivalent locking mechanism.

Do a little reading on lock() and Monitor.Enter()/Monitor.Exit() so you get an idea of when it's appropriate to use synchronization. It's best to add that stuff to the code earlier rather than later as the problems that occur without using synchronization can sometimes be sporadic and pretty hard to track down afterwards.

Actually, as I see it, with two exceptions, the thread only needs to read the state of controls in the UI. I've already taken care of one with an Invoke (something else to learn.) I'll just have to apply the same technique to the other instance.

Thanks very much for the advice.

---

Lilith

.