

Re: AfxBeginThread question

Source: <http://www.tech-archive.net/Archive/VC/microsoft.public.vc.mfc/2008-01/msg00351.html>

- *From:* Joseph M. Newcomer <newcomer@xxxxxxxxxxxxx>
 - *Date:* Tue, 08 Jan 2008 19:31:12 -0500
-

See below...

On Tue, 08 Jan 2008 11:17:46 -0800, Zapanaz <<http://joecosby.com/code/mail.pl>> wrote:

Thanks everybody for the responses, they were all useful.

I would really appreciate your opinions on this though; it seems to me that there isn't really a way to use AfxBeginThread in a way that is both safe and desirable.

Huh? This comes as a serious surprise to those of us who take AfxBeginThread for granted as a fundamental technique we use everywhere. It is highly desirable, and when done properly, is perfectly safe

To use AfxBeginThread, you:

– Implement a "controlling function" on this prototype

```
UINT AddFilesHdThread(LPVOID pParameter)
```

More commonly this should be a static member of your class rather than a global function

– Call AfxBeginThread with the controlling function and a value for pParameter

```
CWinThread * pThreadPointer = AfxBeginThread(AddFilesHdThread,  
&someDataObjectWhichTheThreadNeeds, THREAD_PRIORITY_BELOW_NORMAL,  
NULL, NULL, NULL)
```

Note that the last three parameters default and are rarely written explicitly. Note that

Re: AfxBeginThread question

the stack size is a UINT and therefore "NULL" makes no sense, and the creation flags are a DWORD and therefore NULL makes no sense. These would be written as 0, 0 if you want them 0. The last parameter, which is a pointer, would be written as NULL. Note also that playing with thread priorities is an exercise fraught with risk, and most of us simply default to threads with the same priority. A below-normal thread could end up taking vastly longer than required to do a task because it can be starved.

The system then creates a worker thread, which executes your controlling function, then exits.

Now my usual paradigm for a thread would be

- Parent app creates thread
- On request to exit parent app, parent app notifies thread to close
- Thread has internal logic to shut down safely on notification
- Thread shuts down safely and notifies parent
- Parent closes

That's right

But if I use AfxBeginThread, it creates the thread, it's always a CWinThread, and not a subclass I create.

And your point is what, exactly? Why do you care?

So I can't create some means of receiving a message, as far as I can see, in the created thread, and thereby having a way to accomplish step 2 above.

That's a different question. In the case where you wish to handle messaging, you would create a class *derived* from CWinThread. You would use the *other* form of AfxBeginThread. Let CMyThreadClass be that class. Then you would do

```
CMyThreadClass * thread = (CMyThreadClass *)AfxBeginThread(RUNTIME_CLASS(CMyThreadClass),  
THREAD_PRIORITY_NORMAL,  
0, // default stack size (or specify one)  
CREATE_SUSPENDED);
```

```
thread->SomeValue = SomeParameterOfInterest;  
thread->OtherValue = AnotherParameterValue;  
....  
thread->ResumeThread();
```

Re: AfxBeginThread question

Re: AfxBeginThread question

I could, along the lines of what Scott McPhillips said, do something like:

- Create thread, increment an application-level thread counter
- On thread exit, notify parent
- decrement thread counter

This is classic reference-counting for threads, and you would typically do this whenever you were spinning off multiple threads. Remember my code where I said to test if the thread were running, and clear the thread running flag? The generalization is to use a counter, decrement it to clear it, and the test for thread running is a counter > 0

So if the user tries to exit, if there are still threads running, I could notify the user that I am waiting for threads to complete, and pause or something until the threads all exit, and then exit the application. But that is really undesirable, if I was the user I would find that very annoying.

Pause or something? Probably a Really Bad Idea. I would simply disable the controls I didn't want to have active and return to the message pump. This allows me to continue to handle messages from the thread(s) without having any problems. Also, help works, About works, and any other command/option/etc. you want to have working works. As soon as you say "pause" this suggests you are trying to impose sequential semantics on an asynchronous methodology, and you can assume this is uniformly a losing idea.

But all the ways of shutting the thread down safely depend on being able to notify it.

Read my essays on worker threads and UI threads on my MVP Tips site. A simple BOOL will suffice for a worker thread; not clear why you think something as elaborate as messaging is required to solve a simple problem.

I don't really see any member objects or methods here

[http://msdn2.microsoft.com/en-us/library/f8kx35st\(VS.71\).aspx](http://msdn2.microsoft.com/en-us/library/f8kx35st(VS.71).aspx)

That I could use to tell the thread I want it to stop.

Re: AfxBeginThread question

That's because it is so easy. Set a flag telling the thread to stop.

And anyway, even if I did, it would require having a pointer or handle to the thread itself within the controlling function,

Why? If you are shutting down all the threads, a simple system-wide static that says "everyone shut down" suffices. This is one of the very few times a single static variable makes sense, because it represents system-wide state. Otherwise, pass a pointer to the thread-shutdown object in as one of the fields of your parameter block. You hold onto the parameter block pointer and that's all you need. Note this means the thread itself is not permitted to delete the parameter block when it is finished with it; instead, you have to delete the parameter block when you receive the thread termination notification. It is not uncommon to have the PostMessage put the parameter pointer in the WPARAM or LPARAM field of the thread-finished message

so that from within the controlling function I could check whether the thread had received such a message.

Messaging is not required, or even desirable. You are making a fairly trivial problem unduly complex by assuming messaging is needed

But the controlling function isn't a member of CWinThread.

It doesn't have to be, and in fact I rarely keep the CWinThread pointer around for any reason; it isn't very useful most of the time. Read my essay.

I could pass a pointer to the parent when I call AfxBeginThread, get the pointer to the thread, set that as a public member of the parent,

Why would it have to be a public member of the parent? No reason to make it public; in fact, read my essay where I show how to move from C-space to C++-space in two lines of code.

then I could look at that from the controlling function. That really doesn't seem like a good idea in general (it seems like I can't necessarily depend on the parent having already set that value the first time I try to dereference it in the thread,

Re: AfxBeginThread question

The value has to be set to "don't stop" before you launch the thread

because they are now asynchronous), but in any event I don't see an appropriate method or member.

You're looking for subtle solutions to simple problems.

I guess I had the vague idea that there had to be some safe way of handling this or there wouldn't be an AfxBeginThread(), but that left me with a lot of unanswered questions, which were what I posted before.

Of course there is; I've just pointed out several, and my essay talks about more.

So anyway ... what is you all's opinion? It looks like I should just not use AfxBeginThread().

And how, exactly, do you plan to create threads without using it? Threading isn't all that hard if you follow a few simple guideliness. Read my essays.

joe

Joseph M. Newcomer [MVP]

email: newcomer@xxxxxxxxxxxxx

Web: <http://www.flounder.com>

MVP Tips: http://www.flounder.com/mvp_tips.htm

.