

Re: Thread.Abort()

Source:

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.languages.csharp/2006-10/msg03388.html>

- *From:* mabra <mabra@home>
 - *Date:* Thu, 19 Oct 2006 22:43:24 +0200
-

Hi !

I just went to <http://msdn2.microsoft.com/en-us/library/ty8d3wta.aspx> to verify your statement about the deprecation of this methods, but could not verify.

Additionally, I would leave net immediately; For example, if your mentioned "copy a file to a location", would need too much time, because the underlying system is not responsible, I must have the option to kill that thread and re-try the whole operation at a later moment. Naturally, one has to deal with data consistency in this case very strong.

Just my two cents.
Best regards,
Manfred

Adityanand Pasumarthi wrote:

Hi Alan,

Here is how your class and thread method handler may look like...

```
Class MyThreadWrapper
private _myThread
// (Important: Default state should be signaled, i.e. True)
private _continueEvent System.Threading.ManualResetEvent private enumerated type _state
(pause, stop, start, resume)
method Pause
{ set _state = pause, _continueEvent.Reset() }
method Stop
{ set _state = stop, _continueEvent.Set() }
method Start
{ set _state = start
_myThread.Start()
}
method Resume
{ set _state = resume, _continueEvent.Set() }
procedure DoSomething()
loop
_continueEvent.Wait(-1) // Wait infinitely on the manual reset event
```

Re: Thread.Abort()

```
if (_state == stop) { break; }  
Copy a file to a location  
Update database record  
Read the file content  
Write the content to a log file  
end loop
```

What we are doing here is that the thread loop will just wait for `ManualResetEvent` to be in signaled state (`_continueEvent.Set()`) before continuing with every iteration in the loop. The event will be in this state when the thread first starts. When we call `Pause` the `_continueEvent` will be in non-signaled state and the thread will finish the current loop iteration and then before beginning the next iteration will wait for the event to be in signaled state. When we call `Resume` after some time the `_continueEvent` will be set to signaled state and the thread will start running again.

Calling `Stop()` when we are in running state (start or resume) will cause the loop to terminate after completing its current iteration. Calling `Stop()` when we are in pause state will set the `_continueEvent` to signaled state and immediately since the `_state` is stop, the loop will break and the thread will end safely.

Let me know if this helps.