

## Re: exception handling with events

**Source:**

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.languages.csharp/2004-12/5062.html>

---

**From:** DanGo (*dgolick\_at\_gmail.com*)

**Date:** 12/22/04

Date: 22 Dec 2004 13:14:31 -0800

You must call EndInvoke to avoid memory leaks.

The good news is that any exception that your DoAction threw will be thrown when you call EndInvoke (that's so cool!)

So when should you call EndInvoke?

The second to last argument to BeginInvoke is an AsyncCallback that will be called when your function has completed. You can use this to call EndInvoke.

Of course you need the delegate to fire EndInvoke. The delegate is hidden inside the argument passed to your AsyncCallback method. IAsyncResult does not contain the delegate but it can be safely cast to AsyncResult (in the System.Runtime.Remoting.Messaging namespace) this contains the delegate which you can then use to call EndInvoke.

You should call EndInvoke in a try/catch block to catch any exceptions your asynchronous method threw.

Your AsyncCallback executes on the thread pool thread so don't directly access UI from the callback (use control.BeginInvoke).

Instead of :

```
server.DoAction.BeginInvoke (param1, param2, null, null);
```

use :

```
Server.DoAction.BeginInvome(param1, param2, new  
AsyncCallback(ActionCompleted), null);
```

```
void ActionCompleted(IAsyncResult iar)  
{  
System.Runtime.Remoting.Messaging.AsyncResult ar = iar as  
System.Runtime.Remoting.Messaging.AsyncResult;  
if (iar != null)  
{  
EventHandler eh = ar.AsyncDelegate as EventHandler;
```

```
if (eh != null)
{
try
{
eh.EndInvoke();
}
catch(Exception ex)
{
// the exception thrown by Server.DoAction
// process the exception
// If you want to display it on the ui make sure to use
Control.BeginInvoke since this routine is executing on the thread pool.
}
}
}
```