

Re: How to force the program to continue after unhandled exception detection

Source:

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.languages.csharp/2008-03/msg01275.html>

- *From:* "Peter Duniho" <NpOeStPeAdM@xxxxxxxxxxxxxxxxxxxx>
 - *Date:* Tue, 11 Mar 2008 10:25:51 -0700
-

On Tue, 11 Mar 2008 08:00:25 -0700, nano2k <adrian.rotaru@xxxxxxxxxxxxx> wrote:

Pete,

First of all, thanks for your patience.

You may repro the behavior with VSNET 2005 like this:

1. Create a winform app.

2. Put a button on the form.

3. Add this handler for form's Load event:

```
private void Form1_Load (object sender, EventArgs e) {  
    Application.ThreadException +=  
    new  
    System.Threading.ThreadExceptionHandler(Application_ThreadException);  
}
```

From the MSDN documentation for the Application.ThreadException event, in a clearly emphasized "Note:" box:

"To use this event, you must attach a handler before you call Application.Run"

<http://msdn2.microsoft.com/en-us/library/system.windows.forms.application.threadexception.aspx>

[...]

7. Normally, if no event handler was defined for Application.ThreadException event, the application would crash when the button is clicked.

Run the application without debugger: CTRL-F5. Click the button: the error message appears. Dismiss the message. The application is running normally.

I'm surprised you even got it to work at all using the code you posted.

Re: How to force the program to continue after unhandled exception detection

8. Run the application with the debugger attached. Click the button. The debugger detects the unhandled exception and stops the program at the following line showing the exception helper:

```
throw new ApplicationException("Oops! Unhandled exception!");
```

Cool, until now, it's perfect! Now, try to continue the execution of the program. No chance! No matter what settings you change in Debug/Exceptions section.

I am unable to reproduce your problem, at least when I used the ThreadException event correctly. I haven't bothered to try using it wrong, but I can believe that there might be a difference.

When I set up a ThreadException handler correctly, by subscribing the handler before I call Application.Run(), then once the debugger detects the exception and interrupts the program, I can simply continue execution and the program continues running just as it would have had the debugger not been present.

If you can reproduce the problem you're describing when you've used ThreadException correctly, then there's something for you to be concerned with. But in that case, it means that there's something about your installation that is broken.

Normally it would work just as you desire (and as I noted, you can even disable interrupting execution for handled exceptions altogether...the default is for the debugger to break any time an exception isn't handled by your own code, but you can set it for other behaviors, including breaking only when the exception is unhandled altogether, or breaking whether or not the exception is handled anywhere).

Pete

.