

Re: I finally found the wooden stake to drive through my program's still beating heart!

Re: I finally found the wooden stake to drive through my program's still beating heart!

Source: <http://www.tech-archive.net/Archive/VB/microsoft.public.vb.enterprise/2006-02/msg00028.html>

- *From:* Steve Richfield <Steve@xxxxxxxxxxxxxxxxxxxxxxxx>
 - *Date:* Sun, 05 Feb 2006 21:45:37 -0700
-

Michael,

1. The best way of telling what is wrong is to F8 through the Unload event routine and watch for anything strange. I quickly discovered that on different attempts at unloading, that one or more controls would take ONE FULL MINUTE or more just to execute their .Unregister command!

Why do you need to unregister the control?

100.0% of what I know about this:

Their ActiveX controls have an Unregister method that their documentation says that I should call before exiting.

Do you mean unregister as in the COM unregister or is this something else?

Damned if I know. Is there some way to tell?

Also, why do you need to show any user interface during this time? Couldn't you just hide the form and allow all this to happen in the background as if the app has already terminated?

Hmmm, an interesting idea. Now that the code appears to work OK, then why not! Indeed, perhaps the first statement of all Unload routines should be:

```
Me.Visible = False
```

Thanks.

2. I was executing my Unload statement from way down deep in my code, e.g. as the result of interpreting a command that had been placed on the

Re: I finally found the wooden stake to drive through my program's still beating heart!

Re: I finally found the wooden stake to drive through my program's still beating heart!

clipboard, and plucked off and thrown into a queue when the speech synthesizer threw a Done event. Once I had executed my Unload procedure, the code following the Unload statement sometimes made reference to a control, which would apparently "randomly" restart my form!

Any reference to any control on the form or any property of the form will cause the form to reload. It's not random, it will do this consistently.

No, the mechanism isn't any more random than a "pseudo random" number generator. The "random" component here is the state of my complex state machine at the instant when the user "pulls the plug" on the application.

The cure for this is to set an Unloaded Boolean variable at the end of my Unload routine, and put the following line of code just after every subroutine call that could eventually result in an Unload statement being executed:

```
If Unloaded Then Exit Sub
```

usually that shouldn't be needed.

The only viable options I could see were that or use an End statement to skip rattling back through the complex subroutine structure. See my reply to Bob for more explanation of this.

Remember, exiting does NOT come from some simple button push, but from some AI program considering what you SAID.

3. A form won't go away so long as there are any active timers. I had added a watchdog timer to kick things back into operation if they got hung up for some reason, but forgot to set its Enabled to False in the Unload routine. That timer was keeping my form from terminating, even though it was not actually executing any code at the time I was Unloading.

I've never noticed this to be true, a form should unload fine with timers active.

Perhaps so, but I sure saw a MAJOR difference when I added the

```
Timer3.Enabled = False
```

statement to my Unload procedure. "Mine is not to wonder why ..."

Re: I finally found the wooden stake to drive through my program's still beating heart!

Re: I finally found the wooden stake to drive through my program's still beating heart!

What might be happening here is that timers may get one last tick after the Unload if they were active during the Unload. This would sure explain my form staying up for most of a minute and then going away. Most people use short-fuse timers, which wouldn't present any problem if this is the case. This timer had a one-minute period.

BTW, my application has 3 timers, which are:

1. A 200 ms timer to periodically check the clipboard when my state is reasonable to speak whatever might be on it, i.e. the user is NOT then talking. When the user is talking, I must shut down EVERYTHING to avoid messing up the real-time innards of the speech recognition system.
2. A 2-second timer to recognize long pauses typical of turning the conversation over to the computer.
3. A one-minute watchdog-like timer that runs a small diagnostic whenever it ticks, and if things don't appear to be proceeding normally, it resets the state machine back to an idle condition. I considered the classical sort of a watchdog timer, but they only catch completely dead systems and NOT complex state machines that have run into the rough.

Steve Richfield

.