

Re: loosing messages leaks my app...

Re: loosing messages leaks my app...

Source: <http://www.tech-archive.net/Archive/VC/microsoft.public.vc.mfc/2008-06/msg00846.html>

- *From:* Joseph M. Newcomer <newcomer@xxxxxxxxxxxxx>
 - *Date:* Sun, 22 Jun 2008 21:45:43 -0400
-

On Wed, 18 Jun 2008 19:03:21 +0200, Norbert Unterberg <nunterberg@xxxxxxxxxxxxxxxxxxxxx> wrote:

I see several problems with your design. See below

Norbert

.rhavin grobert schrieb:

hello everyone...

i have a hthread-aware ctrl-class that posts commands from other threads automatically to the gui-thread before executing them. looks like this:

I assume you have one GUI thread and several worker threads with no GUI interaction at all, right? You are using PostMessage which does not post messages to the GUI thread. It posts messages to a window, not to a thread.

And, of course, it would be a serious error to post to the GUI *thread*. This would require PostThreadMessage, which would be erroneous to the main GUI thread.

```
//-----  
// prepares a message  
bool CQControl::_GUIPost(QUAD qMsg, PCVOID pMsg, DWORD dwLen)  
const {  
    SQMsgBlock* pMB = NULL;  
    try {  
        pMB = new SQMsgBlock;  
        pMB->dwLen = dwLen;  
        if (dwLen == NULL)  
            pMB->ptBlock = NULL;  
        else {
```

Re: losing messages leaks my app...

```
pMB->ptBlock = malloc(dwLen);
```

You do not check if malloc was successful.

BTW, you are in the C++ world. Why not make SQMsgBlock a class that takes pMsg, dwLen and qMsg as constructor arguments and frees the ptBlock memory in its destructor? You could remove the GuiPostDelete function.

I find the whole collection of code to be so convoluted as to be almost unreadable. For example, a try/catch block around a piece of code that cannot throw an exception unless it takes an access fault. Use of malloc in a C++ program. Use of memcpy when it shouldn't be required at all.

```
memcpy(pMB->ptBlock, pMsg, dwLen);
}
pMB->qType = qMsg;
pMB->hReturn = 0;
if (_GUIPost(pMB))
return true;
_GUIPostDelete(pMB);
return false;
} catch (...) {
```

What kind of exceptions do you expect here that you want to keep from the caller? Using the general catch(...) form is considered bad practice because it tends to hide bugs.

```
_GUIPostDelete(pMB);
return false;
}
}
```

```
//-----
// post a message to the GUI-thread or execute it if called from GUI-
thread
bool CQControl::_GUIPost(SQMsgBlock* pMB) const {
if (AfxGetThread() == _GUI()) {
// we are the GUI-thread, so we call receive-fn directly...
const_cast<CQControl&>(*this)._GUIReceived(pMB);
```

A const cast of the this pointer ... why do you declare the function const when it is not?

In addition, why don't you always use PostMessage? Making the decision here

Re: losing messages leaks my app...

Re: losing messages leaks my app...

could get you into trouble because it might change the order of message processing. Messages from the same thread are handled before messages in the queue even if they were posted later.

I made this same comment in response to another posting of this same question. This is completely wrong, and I think just silly. I didn't mention the ordering problem, but it is an error, and, for example, I always use PostMessage so I get correct ordering of logging messages

```
return false; // delete block
}
if (_HWND() == 0)
return false;
```

Hmm, why this? Why can the handle be NULL without being an error? If the worker threads rely on the control, then you should not destroy the control before the threads have terminated.

I'm not even sure what _HWND() is! I can't see how a general queuing routine could POSSIBLY know the unique target window for which this is intended.

```
pMB->hControl = _HWND();
```

Why do you need the handle in the message? PostMessage postes the message to the correct window anyway, so it is not needed for message delivery.

```
return (::PostMessage(_HWND(), WM_QWCOMMAND, 0, (LPARAM)
pMB) != 0);
```

What value is WM_QWCOMMAND? If it is "derived" from WM_USER then you can get into trouble if your control is derived from some window control. WM_USER messages are reserved for private window classes. For application-defined messages, use either WM_APP+X or RegisterWindowMessage().

```
}
```

```
//-----
```

Re: losing messages leaks my app...

```
// delete and free a Msg-Block
void CQControl::_GUIPostDelete(SQMsgBlock* pMB) {
    if (pMB == NULL)
        return;
    free(pMB->ptBlock);
    delete pMB;
}
```

```
//-----
// returns true if GUIPost received
bool CQControl::QPreTranslateMessage(MSG* pMsg) {
```

why do you use PreTranslateMessage?

If you want to handle a message, just create a message for your message. It is called whenever the respective message is being processed. No additional checks required.

```
    if (pMsg == NULL)
        return false;
```

Do you think windows would call you with a non-existent message?

If you really need to check the pointer, this would be a candidate for an ASSERT(pMsg);

```
    if (pMsg->message != WM_QWCOMMAND)
        return false;
    SQMsgBlock* pMB = reinterpret_cast<SQMsgBlock*>(pMsg->lParam);
    _GUIReceived(pMB);
    _GUIPostDelete(pMB);
    return true;
}
```

in the CWnd, the fn PreTranslateMessage() calls QPreTranslateMessage() to catch my commands; all runs fine except that i still have some leaks, so even if ::PostMessage() returns that the message was posted, it never gets to the PreTranslateMessage() of the ctrl.

So the question is: is there an application-wide PreTranslateMessage() for the GUI-Thread that i can override to dispatch certain messages myself?

That would be a windows hook. But I still do not understand why you need all this complex code. If your control is your control, then it can just handle the messages on its own, no need to hook something.

I think what you are really looking for is control subclassing.

Re: losing messages leaks my app...

This is the most bizarre code I have ever seen to solve a trivial problem. It implements a poor solution incorrectly, using many times the amount of code actually required to accomplish what is fundamentally trivial.

```
PostMessage(UWM_WHATEVER, (WPARAM)new Thing(..args...));
```

is so much easier to write, and if you want to be really fussy, the correct code is

```
Thing * p = new Thing;
```

```
if(!PostMessage(UWM_WHATEVER, (WPARAM)p))
```

```
delete p;
```

but in fact, if PostMessage fails, there are so many things wrong already that losing a bit of storage is going to be the LEAST of your problems.

joe

Norbert

Joseph M. Newcomer [MVP]

email: newcomer@xxxxxxxxxxxx

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

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

.