Subject: BUG: "Close();" does not!
Posted by slashupp on Mon, 03 Feb 2014 06:38:32 GMT
View Forum Message <> Reply to Message

Using svn 6840 on debian:

The call to Close() in following does not close the window:

```
#include <CtrlLib/CtrlLib.h>

using namespace Upp;

class testclose : public TopWindow
{
public:
 typedef testclose CLASSNAME;
 testclose();

 ArrayCtrl ar;

 void armenu(Bar &bar);
 void DoClose();
};

testclose::testclose()
{
 Title("testclose");
 SetRect(100,100,300,300);

 ar.SetRect(2,2,200,200);
 Add(ar);

 ar.WhenBar = THISBACK(armenu);
}

GUI_APP_MAIN
{
 testclose().Run();
}

void testclose::armenu(Bar &bar)
{
 bar.Add("Close", THISBACK(DoClose));
}

void testclose::DoClose()
{
```

Close();
}

If I call DoClose() from a button.WhenPush(..) it works; from a window menu, it works, but from this popup-menu it does not .. ?

Subject: Re: "Close();" does not!
Posted by Shire on Mon, 03 Feb 2014 09:33:49 GMT
View Forum Message <> Reply to Message

Offtopic. This post is for CtrlLib package.

You call Close() of top window from inner event loop (context menu loop). Top window cannot safely terminate inner event loop.

But you can use this workaround:

PostCallback(THISBACK(Close));

Indeed, this situation must be asserted in library, silent return is never good.

Subject: Re: "Close();" does not!
Posted by slashupp on Mon, 03 Feb 2014 11:09:16 GMT
View Forum Message <> Reply to Message

Quote:But you can use this workaround:

Quote:PostCallback(THISBACK(Close));

I replaced 'Close();' with above call, but it has no effect.

My opinion: the call to Close() should work from anywhere and
any 'housekeeping' to make that work should not be my concern?

Subject: Re: "Close();" does not!
Posted by Oblivion on Mon, 03 Feb 2014 13:22:13 GMT

As Shire already mentioned, and as far as I know, you cannot directly call the TopWindow::Close() from an inner event loop (but it seems relatively safe, since if called from an inner loop, Close() method does nothing and simply returns). I might be wrong, but afaik, buttons, top menu etc. does not have a seperate event loop, they pass their events to their parent (here, TopWindow). On the other hand, popup menu and modal dialogues have an inner loop (you can see they are invoked by methods such as Do(), Run(), Execute())

For example, you could have called Close() if the menu was a direct children of topwindow (a top menu). In this case, you are calling from inside an popup menu of ArrayCtrl (which probably invokes the MenuBar::Execute() method) , so you need to break the top level event loop "explicitly". Use TopWindow::Break().

```
void testclose::DoClose()
{
 // Breaks the TopWindow event loop.
 Break();
}
```

Also, note that TopWindow::Close() method is virtual, you can override it to suit your needs.

Regards.

Subject: Re: BUG: "Close();" does not!
Posted by slashupp on Mon, 03 Feb 2014 14:43:20 GMT

Possible solution:

```
void testclose::DoClose()
{
 static int cb=0;
 if (!cb) { cb=1; SetTimeCallback(10, THISBACK(DoClose), 0); }
 else Close();
}
```

The PostCallBack(..)-function does the above but with a 1 ms delay
and my version just increases that time-out value.

This seems a cleaner way than calling Break() (- the word 'break'
sounds like things won't end nicely

Subject: Re: BUG: "Close();" does not!
Posted by nlneilson on Wed, 05 Feb 2014 02:13:34 GMT
View Forum Message <> Reply to Message

I had considerable problems closing threads from the main.

Having each thread close itself from INSIDE that thread was the only positive and error free method that I found.

Subject: Re: BUG: "Close();" does not!
Posted by slashupp on Wed, 05 Feb 2014 07:50:10 GMT
View Forum Message <> Reply to Message

Quote:
Having each thread close itself from INSIDE that thread was the only positive and error free method that I found.

Considering the above - would the possible solution I suggest be safe?

Subject: Re: BUG: "Close();" does not!
Posted by Shire on Wed, 05 Feb 2014 12:45:39 GMT
View Forum Message <> Reply to Message

Quote:
Having each thread close itself from INSIDE that thread was the only positive and error free method that I found.

This is true, normally, thread must do proper uninitialization by itself. Any outside intervention knows nothing about thread stack objects (and allocated resources) and terminating thread is chance of memory leak.

You can tell thread about exiting via bool flag. There is global flag, managed by Thread::ShutdownThreads, or you can create thread own flag.
