

Modifying Binaries: Making a Window Non-Closeable

by R4ndom on Jul.10, 2012, under Intermediate, Tutorials

Introduction

In this tutorial I will discuss various ways of making a window non-closeable. This can come in handy in a variety of cases. Perhaps you would like to display an inspirational quote in a window, "Just because you are unique, does not mean you are useful", for example. Most users would close this window before really having a chance to think about these words of wisdom. Making the window non-closeable helps solve this problem. Or perhaps you would like to remind a co-worker that there is more to life than work, so send him a game (like "Kill Bunnies With Your Genitalia II") and make it so it won't close, thus reminding him that there are always alternatives to working yourself to death.

Types Of Windows

Because Windows has different types of windows, accomplishing this feat is slightly different, depending on which method was used to create the window. There are basically three types of windows; A normal window, A dialog box, A dialog that is acting as the main window. I will explain these three cases separately, as the process is different for each. I have included a binary representing all three types. Let's start with...

A Normal Window

If you open an app in Olly and do a search for All intermodular calls and see "RegisterClass" or "RegisterClassEx" (providing the app is not packed), then you can assume that this is a probably normal window (meaning not a dialog box). I have included the file "Guru.exe" in this download that uses this type of window. It basically just creates a window and displays a helpful message in it:



Loading the app in Olly and doing a search for intermodular calls gives us the following:

Win32 Programmer's Reference

File Edit Bookmark Options Help

Contents Index Back << >>

RegisterClassEx

Quick Info Overview Group

[Now Supported on Windows NT]

The **RegisterClassEx** function registers a window class for subsequent use in calls to the [CreateWindow](#) or [CreateWindowEx](#) function. The [RegisterClass](#) function does not allow you to set the small icon.

```
ATOM RegisterClassEx(
    CONST WNDCLASSEX *lpwctx    // address of structure with class data
);
```

Parameters

lpwctx
Points to a [WNDCLASSEX](#) structure. You must fill the structure with the appropriate class attributes before passing it to the function.

Return Values

If the function succeeds, the return value is an atom that uniquely identifies the class being registered.

If the function fails, the return value is zero. To get extended error information, call [GetLastError](#).

Remarks

All window classes that an application registers are unregistered when it terminates.

Windows 95: **RegisterClassEx** fails if the **cbWndExtra** or **cbClsExtra** member of the **WNDCLASSEX** structure contains more than 40 bytes.

See Also

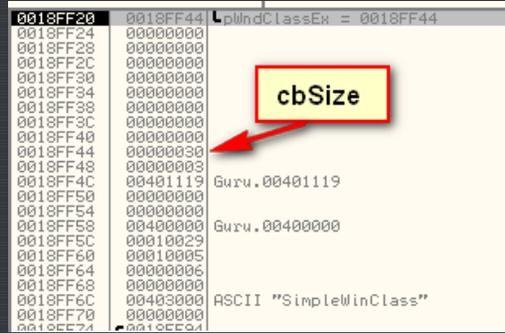
[CreateWindow](#), [CreateWindowEx](#), [GetClassInfoEx](#), [GetClassName](#), [RegisterClass](#), [UnregisterClass](#), [WindowProc](#), [WNDCLASSEX](#)

So RegisterClassExA takes one argument- a pointer to a WNDCLASSEX structure. The preceding PUSH EAX was this pointer. Here is what the API has to say about WNDCLASSEX:

```
typedef struct _WNDCLASSEX {
    UINT  cbSize;
    UINT  style;
    WNDPROC lpfnWndProc;
    int   cbClsExtra;
    int   cbWndExtra;
    HANDLE hInstance;
    HICON  hIcon;
    HCURSOR hCursor;
    HBRUSH hbrBackground;
    LPCTSTR lpszMenuName;
    LPCTSTR lpszClassName;
    HICON  hIconSm;
} WNDCLASSEX;
```

One thing you should know is that if your app calls RegisterClass instead of RegisterClassExA, the WNDCLASS structure will be used instead of the WNDCLASSEX structure. The only difference between the two (that we care about) is the ExA version starts with a cbSize element, whereas the WNDCLASS does not have this element.

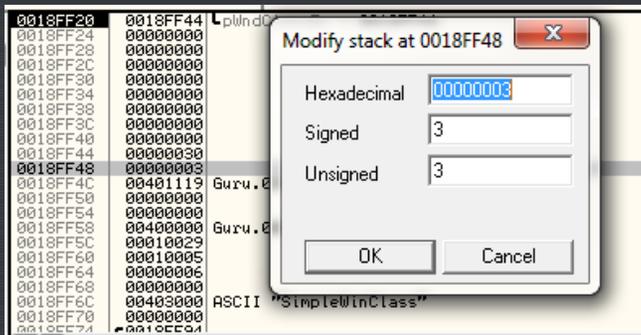
The element of the WNDCLASSEX structure we want is the 'style' element, which is the second argument (the first in WNDCLASS). Let's see this structure for ourselves. Set a BP on the PUSH EAX at address 40109D and re-start the app:



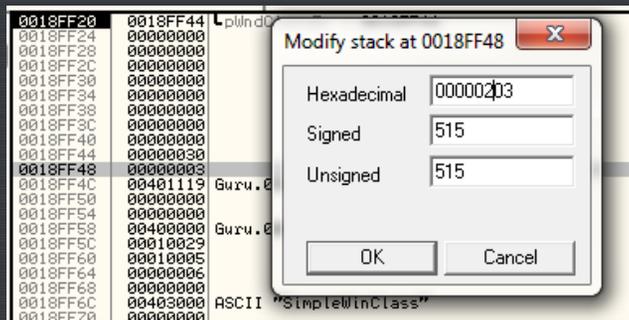
And here is the style argument and the pointer to WndProc. I added the pointer to WndProc because sometimes you want to find where the main WndProc is and this is a good way to do it (though that's material for another tutorial!).



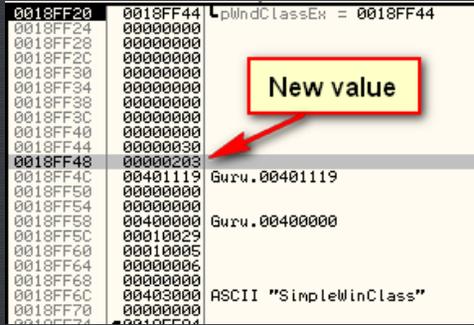
We can see that the style is set to 3. This is the default look of the window and means 'show me a standard window, with all the standard stuff'. Now what we want to do is change this and add a simple command to it that Windows uses to grey out the close button. The constant is CS_NOCLOSE and it disables the Close command on the System menu. CS_NOCLOSE is equal to 0x200. Generally, when you want to combine attributes you OR them together, so that's what we need to do here. We need to OR 0x03 with 0x200 which equals 0x203. Right-click on the stack line at address 18FF48 and choose 'modify':



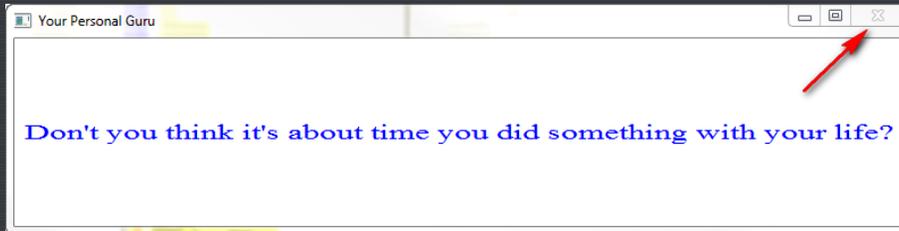
Now let's change this to our new attribute setting:



and click OK. We now have our new setting on the stack:

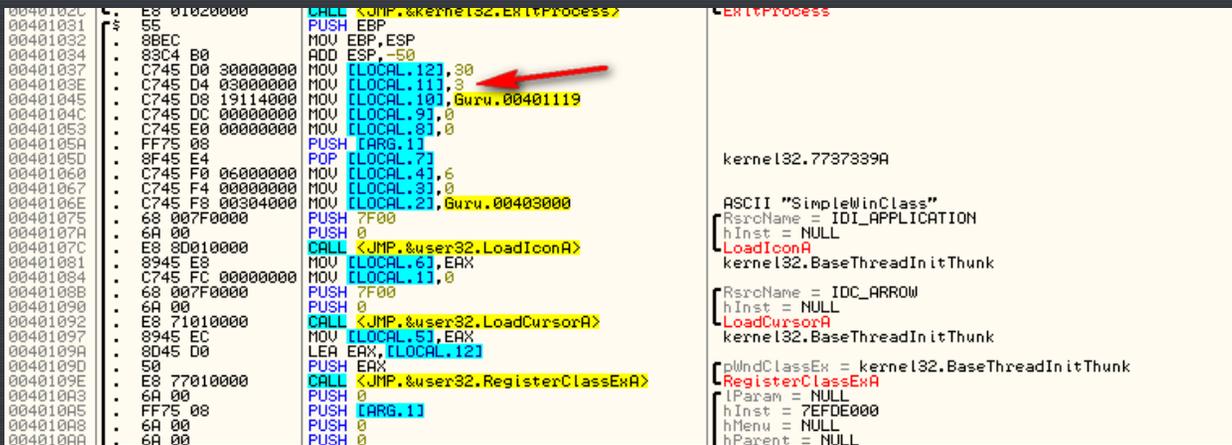


Go ahead and run the app:



and you will notice that you may no longer close this window. Added to the fact that the window can not be covered by another window, it certainly makes it's point. Of course, the user can always minimize it, but that icon just sits there... calling out...wanting to be opened...

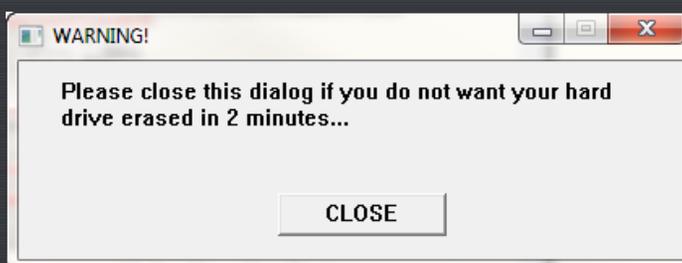
If we want to patch this binary to always do this, we simply need to find where this style value of 3 is being pushed on to the stack and replace it with a 0x203. Looking up a couple lines we can clearly see the culprit:



So patching this line to MOV [LOCAL.11], 203 will patch the app for good 😊.

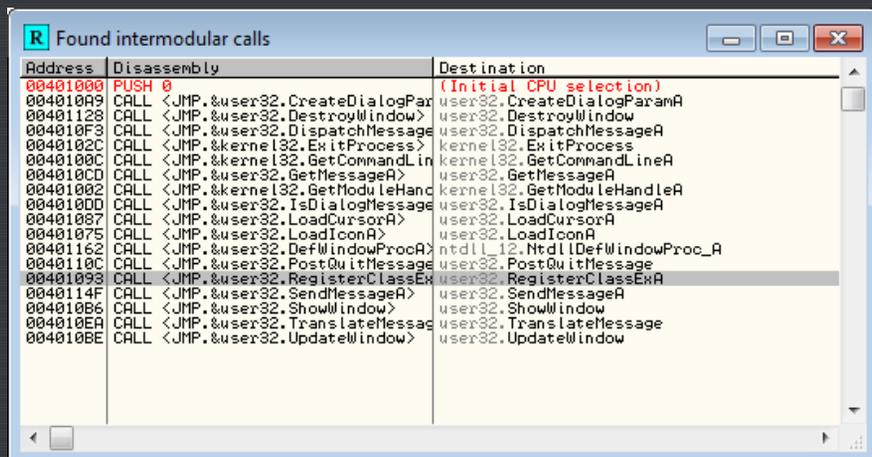
A Dialog Box As Main

Sometimes apps are set up where the main screen is in fact a dialog box. This is helpful in that you can allow Windows to handle a lot of the overhead having to do with buttons, edit boxes etc. Go ahead and load "Helpful.exe" into Olly. Run it and you will see a dialog box:

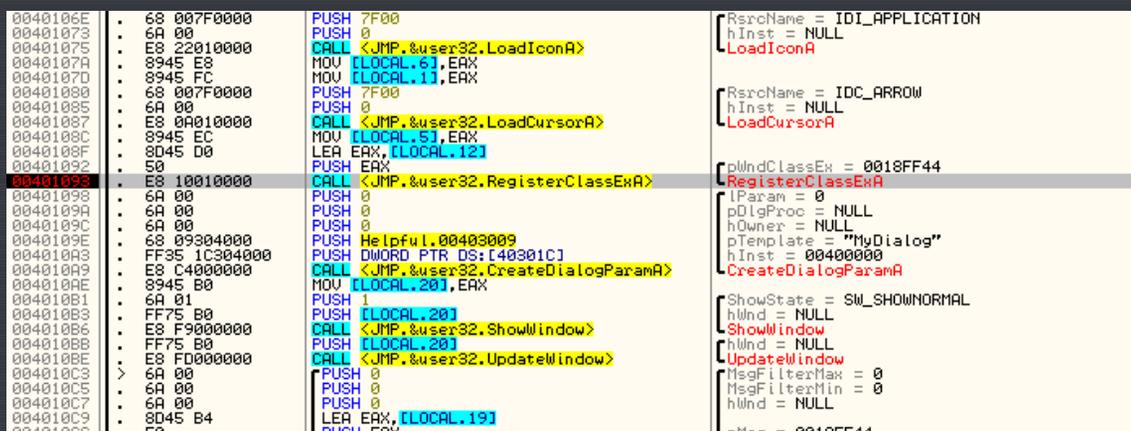


Notice that clicking the CLOSE button or the 'X' on the title bar both close the dialog.

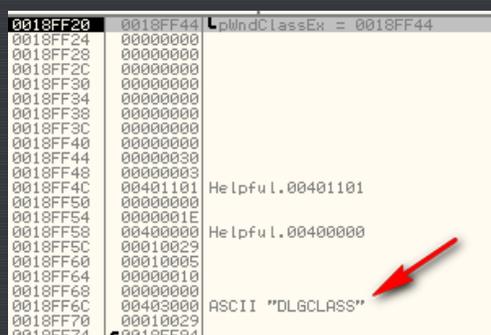
Doing a search in intermodular calls shows that this app has a RegisterClassExA (RegisterClassExA), so we know it sets up it's own window (as opposed to a true dialog opened by another window which will not have a register class, as we will see...):



We will start the same way we did with the previous example. Go to the RegisterClassExA and place a BP on it and re-start the app:

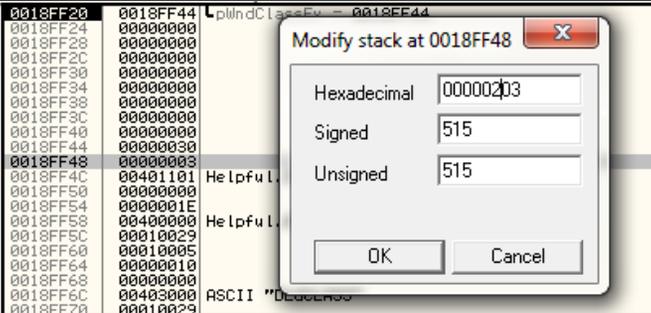


Now look at the stack window:



Notice that most everything is the same, except this time there is an argument called "DLGCLASS". This is to tell windows that this dialog should be our main program window and to send messages to it just like a normal window.

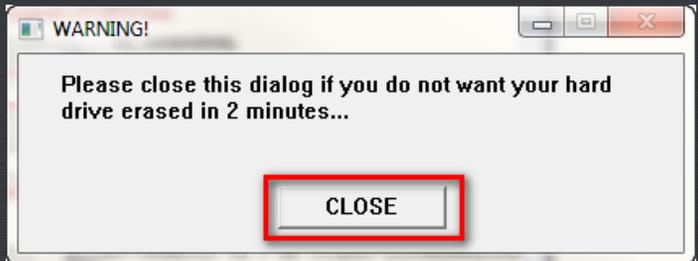
Go ahead and modify the stack and turn the 3 into a 203 so we can grey out the close button:



You can also patch the instruction at address 401203 to always load in 203 as our value:

0040102C	E8 95010000	CALL <JMP.&kernel32.ExitProcess>	ExitProcess
00401031	55	PUSH EBP	
00401032	8BEC	MOV EBP,ESP	
00401034	83C4 B0	ADD ESP,-B0	
00401037	C745 D0 30000000	MOV [LOCAL_12],30	
0040103E	C745 D4 03020000	MOV DWORD PTR SS:[EBP-2C],203	
00401045	C745 DC 00000000	MOV [LOCAL_10],Helpful.00401101	
0040104C	C745 DC 00000000	MOV [LOCAL_9],8	
00401053	C745 E0 1E000000	MOV [LOCAL_8],1E	
0040105A	FF75 08	PUSH [ARG_1]	Helpful.00400000
0040105D	8F45 E4	POP [LOCAL_7]	0018FF44
00401060	C745 F0 10000000	MOV [LOCAL_4],10	
00401067	C745 F8 00304000	MOV [LOCAL_2],Helpful.00403000	ASCII "DLGCLASS"

Typically with dialog boxes there is a button that will close the dialog along with the normal 'X' on the title bar. Sometimes it's called 'OK' or 'Cancel', or in our case it's called 'Close':



What we need to do is get rid of the functionality that allows the dialog to be closed by a button. The easiest way to do this is to find the DestroyWindow call by looking in our intermodular calls again:

Address	Disassembly	Destination
00401123	JNZ SHORT Helpful.0040112F	(Initial CPU selection)
004010A9	CALL <JMP.&user32.CreateDialogParam>	user32.CreateDialogParam
00401128	CALL <JMP.&user32.DestroyWindow>	user32.DestroyWindow
004010F3	CALL <JMP.&user32.DispatchMessageA>	user32.DispatchMessageA
0040102C	CALL <JMP.&kernel32.ExitProcess>	kernel32.ExitProcess
0040100C	CALL <JMP.&kernel32.GetCommandLineA>	kernel32.GetCommandLineA
004010CD	CALL <JMP.&user32.GetMessageA>	user32.GetMessageA
00401002	CALL <JMP.&kernel32.GetModuleHandleA>	kernel32.GetModuleHandleA
004010DD	CALL <JMP.&user32.IsDialogMessage>	user32.IsDialogMessage
004010S7	CALL <JMP.&user32.LoadCursorA>	user32.LoadCursorA
00401075	CALL <JMP.&user32.LoadIconA>	user32.LoadIconA
00401162	CALL <JMP.&user32.DefWindowProcA>	ntdll.12:NtdllDefWindowProc_A
0040110C	CALL <JMP.&user32.PostQuitMessage>	user32.PostQuitMessage
00401093	CALL <JMP.&user32.RegisterClassExA>	user32.RegisterClassExA
0040114F	CALL <JMP.&user32.SendMessageA>	user32.SendMessageA
004010B6	CALL <JMP.&user32.ShowWindow>	user32.ShowWindow
004010EA	CALL <JMP.&user32.TranslateMessage>	user32.TranslateMessage
004010BE	CALL <JMP.&user32.UpdateWindow>	user32.UpdateWindow

Notice that right before the DestroyWindow call, there is a JNZ instruction that allows the program to skip this call.

```

00401101 . 55 1000 PUSH EBP
00401102 . 8BEC MOV EBP,ESP
00401104 . 837D 0C 02 CMP [ARG_2],2
00401108 . 75 09 JNZ SHORT Helpful.00401113
0040110A . 6A 00 PUSH 0
0040110C . E8 91000000 CALL <JMP.&user32.PostQuitMessage>
00401111 . EB 58 JMP SHORT Helpful.0040116B
00401113 > 817D 0C 11010000 CMP [ARG_2],111
0040111A . 75 3A JNZ SHORT Helpful.00401156
0040111C . 8B45 10 MOV EAX,[ARG_3]
0040111F . 837D 14 00 CMP [ARG_4],0
00401123 . 75 0A JNZ SHORT Helpful.0040112F ←
00401125 . FF75 08 PUSH [ARG_1]
00401128 . E8 51000000 CALL <JMP.&user32.DestroyWindow> ←
0040112D . EB 3C JMP SHORT Helpful.0040116B
0040112F > 8B55 10 MOV EDX,[ARG_3]
00401132 . C1EA 10 SHR EDX,10
00401135 . 66:0BD2 OR DX,DX
00401138 . 75 1A JNZ SHORT Helpful.00401154
0040113A . 66:3D BA0B CMP AX,0BBA
0040113E . 75 14 JNZ SHORT Helpful.00401154
00401140 . 6A 00 PUSH 0

```

If a button is pressed that sends a quit message, this call will not jump and DestroyWindow will be called. So what we want to do is make it ALWAYS jump so that DestroyWindow is never called:

```

00401111 . EB 58 JMP SHORT Helpful.0040116B
00401113 > 817D 0C 11010000 CMP [ARG_2],111
0040111A . 75 3A JNZ SHORT Helpful.00401156
0040111C . 8B45 10 MOV EAX,[ARG_3]
0040111F . 837D 14 00 CMP [ARG_4],0
00401123 . EB 0A JMP SHORT Helpful.0040112F
00401125 . FF75 08 PUSH [ARG_1]
00401128 . E8 51000000 CALL <JMP.&user32.DestroyWindow>
0040112D . EB 3C JMP SHORT Helpful.0040116B
0040112F > 8B55 10 MOV EDX,[ARG_3]
00401132 . C1EA 10 SHR EDX,10
00401135 . 66:0BD2 OR DX,DX

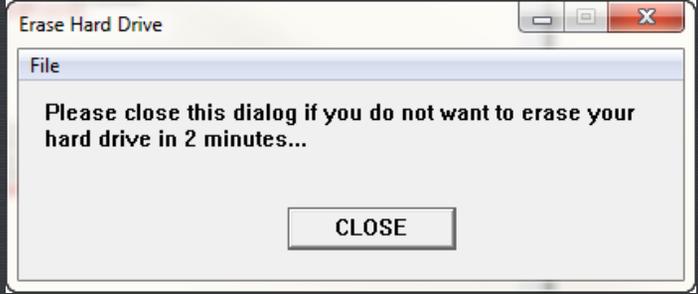
```

Here, I just hit the space bar on the line and changed the JNZ to a JMP. Now, any button on the dialog that tries to close the dialog window through a button will fail, as it will always jump over the call to DestroyWindow. Go ahead and try it and you will notice that the close button does nothing and the 'X' in the title bar is greyed out.



A Standard Dialog Box

A standard dialog box is usually called from another window (think of the "Are you sure" dialogs with the OK and CANCEL button). Though this is the norm, they do not always have to be called from another window. The "Helpful2.exe" binary simply opens a dialog with no parent, though the techniques on this dialog will work for dialogs called from another window as well. Go ahead and run the app:



You will notice that it looks almost the same as the last one, except this time there's a menu added:

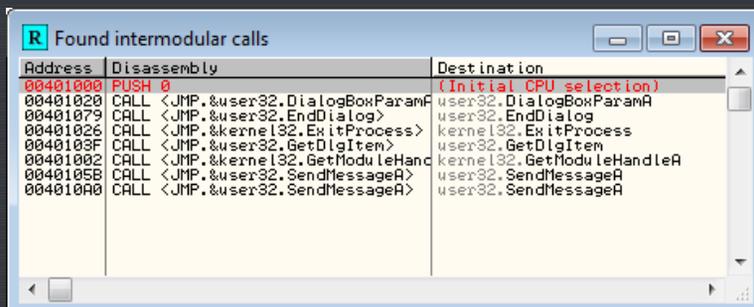


I wanted to add this to show you how to defeat menu items as well. Loading the app in Olly is where you'll notice the difference though:

Address	Disassembly	Comment
00401000	PUSH 0	
00401002	CALL <JMP.&kernel32.GetModuleHandleA>	[GetModuleHandleA
00401007	MOV DWORD PTR DS:[403014],EAX	kernel32.BaseThreadInitThunk
0040100C	PUSH 0	[lParam = NULL
0040100E	PUSH Helpful2.0040102B	DlgProc = Helpful2.0040102B
00401013	PUSH 0	hOwner = NULL
00401015	PUSH Helpful2.00403000	pTemplate = "MyDialog"
0040101A	PUSH DWORD PTR DS:[403014]	hInst = NULL
00401020	CALL <JMP.&user32.ShowDialogBoxParamA>	[DialogBoxParamA
00401025	POP EAX	ExitCode = 77373388
00401026	CALL <JMP.&kernel32.ExitProcess>	[ExitProcess
0040102B	PUSH EBP	
0040102E	MOV EBP,ESP	
00401032	CMP [ARG_2],110	
00401035	JNZ SHORT Helpful2.00401046	[ControlID = BBB (3003.)
00401037	PUSH 0BBB	hWnd = 7EFDE000
0040103C	PUSH [ARG_1]	[GetDlgItem
0040103F	CALL <JMP.&user32.GetDlgItem>	
00401044	JMP SHORT Helpful2.004010B0	
00401046	CMP [ARG_2],10	
0040104A	JNZ SHORT Helpful2.00401062	
0040104C	PUSH 0	[lParam = 0
0040104E	PUSH 7D02	wParam = 7D02
00401053	PUSH 111	Message = WM_COMMAND
00401058	PUSH [ARG_1]	hWnd = 7EFDE000
0040105B	CALL <JMP.&user32.SendMessageA>	[SendMessageA
00401060	JMP SHORT Helpful2.004010B0	
00401062	CMP [ARG_2],111	
00401069	JNZ SHORT Helpful2.004010A7	
0040106B	MOV EAX,[ARG_3]	
0040106E	CMP [ARG_4],0	
00401072	JNZ SHORT Helpful2.004010B0	
00401074	PUSH 0	[Result = 0
00401076	PUSH [ARG_1]	hWnd = 7EFDE000
00401079	CALL <JMP.&user32.EndDialog>	[EndDialog
0040107E	JMP SHORT Helpful2.004010B0	
00401080	MOV EDI,[ARG_3]	
00401083	SHR EDI,10	
00401086	OR EDI,EDI	
00401089	JNZ SHORT Helpful2.004010A5	
0040108B	CMP AX,0BBA	
0040108F	JNZ SHORT Helpful2.004010A5	
00401091	PUSH 0	[lParam = 0
00401093	PUSH 7D02	wParam = 7D02
00401098	PUSH 111	Message = WM_COMMAND
0040109D	PUSH [ARG_1]	hWnd = 7EFDE000
004010A0	CALL <JMP.&user32.SendMessageA>	[SendMessageA
004010A5	JMP SHORT Helpful2.004010B0	
004010A7	MOV EAX,0	
004010AC	C9	LEAVE
004010AD	C2 1000	RETN 10
004010B0	B8 01000000	MOV EAX,1
004010B5	C9	LEAVE

There is very little code. Notice that a window is not created like it was in the last two examples. There is no WNDCLASS and no call to RegisterClass. This looks more like the code you would see in a bigger app when the app decides to open a dialog box. Because there is a parent window already created, you do not have to go through the process of creating a whole new one from scratch.

If you do a search for intermodular calls you will notice that there is no RegisterClass, so changing the WNDCLASS structure to add a NOCLOSE attribute is out of the question:



Fortunately, there is a very simple way to stop not only the 'X' in the title bar, but also any child that tries to close the window, including buttons and menu items. Even though this binary does not have a RegisterClass, it does have an EndDialog. EndDialog is called whenever the program wants to close a standard dialog window. All we have to do is make sure this method is never called.

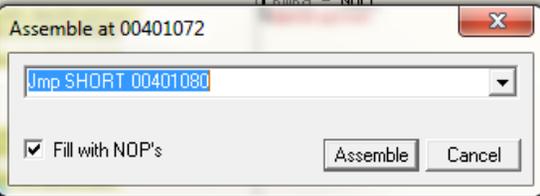
```

0040105B . E8 6C000000 CALL <JMP.&user32.SendMessageA>
00401060 . EB 4E JMP SHORT Helpful2.00401060
00401062 > 817D 0C 11010000 CMP [ARG.2],111
00401069 . 75 3C JNZ SHORT Helpful2.004010A7
0040106B . 8B45 10 MOV EAX,[ARG.3]
0040106E . 837D 14 00 CMP [ARG.4],0
00401072 > 75 0C JNZ SHORT Helpful2.00401080
00401074 . 6A 00 PUSH 0
00401076 . FF75 08 PUSH [ARG.1]
00401079 . E8 42000000 CALL <JMP.&user32.EndDialog>
0040107E . EB 30 JMP SHORT Helpful2.00401080
00401080 > 8B55 10 MOV EDX,[ARG.3]
00401083 . C1EA 10 SHR EDX,10
00401086 . 66:0BD2 OR DX,DX
00401089 . 75 1A JNZ SHORT Helpful2.004010A5
00401093 . 66:0B00 CMP DX,0
    
```

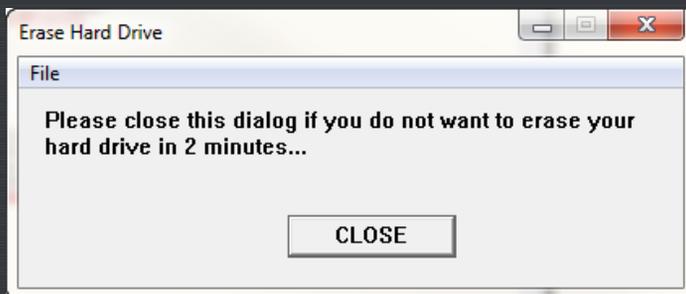
Just like in the last example, there is a JNZ right before the call to EndDialog that, if equal, will allow falling through to the EndDialog call. What we want to do is simply change this JNZ to a JMP like we did in the last example:

```

00401037 . 68 B0B00000 PUSH 0BBB
0040103C . FF75 08 PUSH [ARG.1]
0040103F . E8 82000000 CALL <JMP.&user32.SendMessageA>
00401044 . EB 6A JMP SHORT Helpful2.00401080
00401046 > 837D 0C 10 CMP [ARG.2],10
0040104A . 75 16 JNZ SHORT Helpful2.00401080
0040104C . 6A 00 PUSH 0
0040104E . 68 027D0000 PUSH 7D02
00401053 . 68 11010000 PUSH 111
00401058 . FF75 08 PUSH [ARG.1]
0040105B . E8 6C000000 CALL <JMP.&user32.SendMessageA>
00401060 . EB 4E JMP SHORT Helpful2.00401060
00401062 > 817D 0C 11010000 CMP [ARG.2],111
00401069 . 75 3C JNZ SHORT Helpful2.004010A7
0040106B . 8B45 10 MOV EAX,[ARG.3]
0040106E . 837D 14 00 CMP [ARG.4],0
00401072 > 75 0C JNZ SHORT Helpful2.00401080
00401074 . 6A 00 PUSH 0
00401076 . FF75 08 PUSH [ARG.1]
00401079 . E8 42000000 CALL <JMP.&user32.EndDialog>
0040107E . EB 30 JMP SHORT Helpful2.00401080
00401080 > 8B55 10 MOV EDX,[ARG.3]
00401083 . C1EA 10 SHR EDX,10
00401086 . 66:0BD2 OR DX,DX
    
```



Go ahead and run the app and you will notice the dialog cannot be closed.



-Now go out and do some good.

R4ndom