AddMenu 0.3.0

Enclosed is the source code for AddMenu. It is written in C and was developed using Borland Turbo C/C++ for Windows. The code employs several often discussed, but rarely demonstrated aspects of the Windows API.

Notably, the program processes "hooks". Hooks are ways that you can have Windows send various messages to your application regardless of which application is currently running. This program uses two types, WH_CALLWNDPROC and WH_GETMESSAGE. These are loosely documented in various manuals and online .HLP files, but hopefully this code will help you better understand how to employ these in your own programs.

This program uses hooks in order to add options to the system menus of Windows. Adding options to a given application's menus is trivial, but it is more complicated if you wish to add options to all of the system menus of all of the windows of all applications. Basically, you need some way of intercepting all other application's handling of menus in order to process them yourself. Using hooks is one technique. Basically, I use the WH_CALLWNDPROC to intercept the WM_INITMENU call of all programs in order to ensure that I can add the options I want. I use the WH_GETMESSAGE hook to intercept the WM_SYSCOMMAND call so that I can execute the appropriate commands when a user selects something from a menu.

This is confusing: Why use 2 different hooks to capture different messages. This is because certain hooks only trap certain messages. So, as far as I could tell, of all the possible hooks, only WH_CALLWNDPROC captured the WM_INITMENU call. And only WH_GETMESSAGE trapped all the WM_SYSCOMMAND calls (WH_SYSMSGFILTER traps some WM_SYSCOMMAND messages, but not the relevant ones here).

So, you can examine this code from the point of view of learning about hooks. I don't claim that the code is very well written. It's just one of the few publicly available hook examples that I know of.

For those of us interested in adding options to system menus, this code presents more questions than solutions. Granted, it appears to be a working implementation, but the problem is that it uses WH_CALLWNDPROC. This reportedly degrades system performance. This is understandable: it is passing the lion's share of messages flowing in Windows through an additional interface: your program.

The purpose of the rest of this document is to initiate a discussion of this topic. Those of you uninterested in changing system menus globally can stop reading here. The question for the rest of us is how to manipulate system menus without using WH_CALLWNDPROC. There are several alternatives:

1. Just stick with WH_CALLWNDPROC and live with any performance degradation. (Does anyone have any measures of the degree of degradation?)

2. Figure out some way of using one of the other hooks to trap some other message which can be intercepted and allow us to add the appropriate options to the menu. My experimentation in this respect was of limited success: I was able to add options, but when the menu was painted the first time, they were not being painted correctly (all new options were appearing under "Restore"). If there was some Windows api call to tell Windows to recalculate the appearance of a menu, then all would be well. I haven't found any such call. (Note that a commercial package, WordStar's American Heritage Dictionary (c) suffers similar problems under certain circumstances, but resolve it in others.)

3. Simply subclass the system menu. You can't, unfortunately, do this.

4. Take advantage of Window or Class subclassing (global or otherwise). This topic, like hooks, suffers from a lack of documentation, but Microsoft did publish a technical report document discussing this. (I believe that wuarchive.wustl.edu and ftp.uu.net both have copies.) The idea is to abandon the use of

hooks to process menus, but rather to provide your own callback function for windows/classes which will: a) process any appropriate system menu changes when the right messages are sent to the window; or b) call the window's/classes' original GWL_WNDPROC for all other messages.

Personally, I see several problems with this approach. First, since you can't subclass classes that will be defined in the future, you presumably still need to use a hook to subclass classes/windows that will be defined in the future. Second, you need to keep maintain a table of the GWL_WNDPROC's for all subclassed windows/classes (and I hate maintaining dynamic lists). Third, if we're subclassing all classes/windows that use system menus, is this really that more efficient than hooks. Fine, presumably you wouldn't be intercepting messages to windows without system menus (if you write your code well enough), but you'd still be intercepting all messages to all windows that do have menus. I would think that clever hook programming (avoiding WH_CALLWNDPROC) would be more efficient. Fourth, my understanding is that you run into problems when you subclass windows that are already subclassed (or if some other application subclasses your subclassing). Apparently Windows doesn't keep track of subclassing like it does clipboard chains or hooks, and if a program which subclasses that are subsequently subclasses, attempts to remove itself can cause UAEs (because the program has no way of notifying other apps which are subclassing and calling it's callback function that it is removing itself from memory).

But, clearly I'm missing some easy subclassing approach, because it seems that it is often touted as the best approach for modifying system menus. I don't see it unless there was some single base class that could be subclassed.

If anyone has any thoughts on this topic, I'd love to hear them. If people want to be updated on the conclusions, please let me know. I presume that comp.window.ms.programmer would be the best forum for this discussion, but if there are those of you who don't follow that list can send me your e-mail addresses and I'll summarize (if there's anything to summarize).

Rob Ryan, April 28, 1992
internet: Robert_Ryan@Brown.Edu or st802200@BrownVM.Brown.Edu
bitnet: ST802200@BROWNVM.BITNET
Compu$erve: 70324,227