Rescuing thread messages from modal loops via message filters

devblogs.microsoft.com/oldnewthing/20050428-00

April 28, 2005

Raymond Chen

As we have seen recently, thread messages are eaten by modal loops because they have nowhere to go when dispatched. However, there is a way to see them before they vanish, provided the modal loop is cooperative.

The WH_MSGFILTER message hook allows you to receive messages passed to the `CallMsgFilter` function. Fortunately, all the modal loops in the window manager use `CallMsgFilter` to allow the thread to capture thread messages before they are lost. Therefore, this gives you a way to snoop on messages as they travel through modal loops.

Let's add a message filter to the program we wrote last time to see how messages pass through a message filter. Note that **this is the wrong way to solve the problem**. The correct solution was illustrated last time. I'm doing it the wrong way to illustrate message filters since they are not well-understood. (For example, a valid reason for a message filter would to prevent the menu loop from seeing certain input.)

Start with the program from last the before we changed the `PostThreadMessage` to a `PostMessage`, then make the following changes:

```
HHOOK g_hhkMSGF;
LRESULT CALLBACK MsgFilterProc(int code, WPARAM wParam, LPARAM lParam)
{
 MSG* pmsg = (MSG*)lParam;
 if (code >= 0 && IsThreadMessage(pmsg)) return TRUE;
 return CallNextHookEx(g_hhkMSGF, code, wParam, lParam);
}
BOOL
OnCreate(HWND hwnd, LPCREATESTRUCT lpcs)
{
 g_hhkMSGF = SetWindowsHookEx(WH_MSGFILTER, MsgFilterProc,
    NULL, GetCurrentThreadId());
 if (!g_hhkMSGF) return FALSE;
 DWORD dwThread;
 HANDLE hThread = CreateThread(NULL, 0, ThreadProc,
       UintToPtr(GetCurrentThreadId()), 0, &dwThread);
 ...
```

Here, we installed a message filter hook on our thread so that we can seem messages as they pass through modal loops. The `code` parameter tells us what type of modal loop retrieved the message; we ignore it here since we want to do our filtering for all modal loops.

Run this program and observe that the beeps are no longer lost because our message filter is getting a chance to see them and react to them.

The message filter trick relies on all modal loops sending the messages they retrieve through a message filter before dispatching them. If you are writing code that is going into a library, and you have a modal loop, then you too should call the message filter before dispatching messages you've retrieved, in case the program using your library wants to do something with the message.

```
MSG msg;
while (GetMessage(&msg, NULL, 0, 0)) {
 if (!CallMsgFilter(&msg, MSGF_MYLIBRARY)) {
  TranslateMessage(&msg);
  DispatchMessage(&msg);
 }
}
```

The value `MSGF_MYLIBRARY` is an arbitrary positive value you can choose and document in your library's header file. You can see examples of this in the `commctrl.h` header file:

```
#define MSGF_COMMCTRL_BEGINDRAG     0x4200
#define MSGF_COMMCTRL_SIZEHEADER    0x4201
#define MSGF_COMMCTRL_DRAGSELECT    0x4202
#define MSGF_COMMCTRL_TOOLBARCUST   0x4203
```

These are the message filters called by the modal loops in the shell common controls library.

One question you might ask is, “Why use a message filter hook instead of a `GetMessage` hook?”

Message filter hooks are less expensive than `GetMessage` hooks because they are called only upon request, as opposed to a `GetMessage` hook which is called for every retrieved message. Message filter hooks also tell you **which** modal loop is doing the filtering, in case you want to adjust your behavior accordingly.

The downside of message filter hooks is that all modal loops need to remember to call `CallMsgFilter` as part of their dispatch loop.

Raymond Chen

Follow