Psychic debugging: Why messages aren't getting processed by your message pump

devblogs.microsoft.com/oldnewthing/20140117-00

January 17, 2014

Raymond Chen

The second parameter to the `GetMessage` is an optional window handle that is used to tell the `GetMessage` function to retrieve only messages that belong to the specified window.

A filtered `GetMessage` is nearly always a bad idea, because your program will not respond to messages that don't meet the filter. Unlike a filtered `PeekMessage` (which simply returns "no messages satisfy the filter"), `GetMessage` blocks your thread and does not return until a satisfactory message arrives. Instead, they just pile up like newspapers on your doorstep.

A common mistake I encounter is using a filtered `GetMessage` as the main message pump:

```
hwnd = CreateWindow(...);
if (hwnd == NULL) { return error }
while (GetMessage(&msg, hwnd, 0, 0)) {
 ...
}
```

I don't know for sure, but I'm guessing that the author said, "Well, I created only one window, so clearly that is the only window that can receive messages, and therefore that is the only window I care about."

That may be the only window you explicitly created *in that function*, but there are still plenty of opportunities for other windows to get created. For example, there may be child windows of your main window. Or there may be hidden windows created by other components such as OLE which are used for cross-thread communication. Filtering your message pump's `GetMessage` prevents those other windows from receiving queued messages, and consequently prevents those windows from getting done whatever it was you asked them to do.

When a support request comes in for a program that hangs or acts erratically, you don't think to look at the message pump, because that is nearly always just boilerplate code. Only when you glance at it and notice that the boilerplate code has been tweaked do you realize that the tweaking is the source of the problem. (And when I point out the mistake, I may get a "Thank

you" and possibly even a "I didn't realize that", but never a "This is what I was thinking when I wrote that in the first place," so I never figure out why they went to the extra effort of adding a `GetMessage` filter.)

Armed with this new psychic power, you can help this customer out:

I can’t get combo boxes to work outside of a dialog box. When used as a standalone window, the combo box doesn’t work correctly. It doesn’t respond to mouse hover, sometimes it ignores clicks, sometimes it makes my app hang when I select an item with the mouse. But if I put the combo box inside a dialog, then it works perfectly. As you can see in the attached project, the exact same function (`CreateCombo`) works if called from a dialog box, but not from a regular window. Is there something special about combo boxes that prevent them from being used outside of a dialog box?

```
void CreateCombo(HWND hwndParent)
{
 HWND hwndCombo = CreateWindow(TEXT("combobox"), 0,
  WS_BORDER | WS_CHILD | WS_VISIBLE | CBS_DROPDOWNLIST,
  10, 10, 200, 200, hwndParent, NULL, g_hinst);
 ComboBox_AddString(hwndCombo, TEXT("Item 0"));
 ComboBox_AddString(hwndCombo, TEXT("Item 1"));
 ComboBox_AddString(hwndCombo, TEXT("Item 2"));
 ComboBox_AddString(hwndCombo, TEXT("Item 3"));
 ComboBox_AddString(hwndCombo, TEXT("Item 4"));
 ComboBox_AddString(hwndCombo, TEXT("Item 5"));
 ComboBox_AddString(hwndCombo, TEXT("Item 6"));
 ComboBox_AddString(hwndCombo, TEXT("Item 7"));
 ComboBox_AddString(hwndCombo, TEXT("Item 8"));
 ComboBox_AddString(hwndCombo, TEXT("Item 9"));
}
// Dialog box version
INT_PTR CALLBACK DialogProc(
    HWND hdlg, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
 switch (uMsg) {
 case WM_INITDIALOG:
  CreateCombo(hdlg);
  return TRUE;
 case WM_CLOSE:
  EndDialog(hdlg, 0);
  return TRUE;
 }
 return FALSE;
}
void TestDialog()
{
 DialogBox(g_hinst, MAKEINTRESOURCE(IDD_DIALOG),
           NULL, DialogProc);
}
// Plain window version
LRESULT CALLBACK WndProc(
    HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
 switch (uMsg) {
 case WM_CREATE:
  CreateCombo(hwnd);
  return 0;
 case WM_DESTROY:
  PostQuitMessage(0);
  return 0;
 }
 return DefWindowProc(hwnd, uMsg, wParam, lParam);
}
void TestWindow()
{
 WNDCLASS wc = { 0, WndProc, 0, 0, g_hinst, NULL, NULL,
```

```
                (HBRUSH)(COLOR_WINDOW+1), NULL, TEXT("Test"));
 RegisterClassEx(&wc); // succeeds
 HWND hwnd = CreateWindow(TEXT("Test"), TEXT("Test"),
                  WS_OVERLAPPEDWINDOW | WS_VISIBLE | WS_CLIPCHILDREN,
                  CW_USEDEFAULT, CW_USEDEFAULT,
                  CW_USEDEFAULT, CW_USEDEFAULT,
                  NULL, NULL, g_hinst, NULL);
 MSG msg;
 while (GetMessage(&msg, hwnd, 0, 0)) {
  TranslateMessage(&msg);
  DispatchMessage(&msg);
 }
 UnregisterClass(TEXT("Test"), g_hinst);
}
```

Raymond Chen

Follow