A different type of dialog procedure

devblogs.microsoft.com/oldnewthing/20031112-00

November 12, 2003

Raymond Chen

In the discussion following my entry about dialog procedure return values, somebody suggested an alternate dialog design where you just call `DefDlgProc` to do default actions (the same way you write window procedures and `DefWindowProc`) rather than returning TRUE/FALSE.

So let's do that. In fact, we're going to do it twice. I'll cover one method today and cover an entirely different method later this week. Each method consists of a simple kernel of an idea; the rest is just scaffolding to make the kernel work.

The first way uses a recursive call from the dialog procedure back into `DefDlgProc` to trigger the default behavior. This technique requires that you have a flag that lets you detect (and therefore break) the recursion. Since you typically have instance data attached to your dialog box anyway, it's not too hard to add another member to it.

The kernel is to "subvert the recursive call". `DefDlgProc` calls your dialog procedure to see what you want to do. When you want to do the default action, just call `DefDlgProc` recursively. The inner `DefDlgProc` will call your dialog procedure to see if you want to override the default action. Detect this recursive call and return FALSE ("do the default"). The recursive `DefDlgProc` will then perform the default action and return its result. Now you have the result of the default action, and you can modify it or augment it before returning that as the result for the dialog box procedure, back to the outer `DefDlgProc` which returns that value back as the final message result.

Here's the flow diagram, for those who prefer pictures:

```
Message delivered
-> DefDlgProc
   -> your dialog procedure
      decide what to do
      want to do the default action
      -> DefDlgProc
         -> your dialog procedure
            detect recursion
         <- return FALSE
         DefDlgProc sees FALSE
         performs default behavior
      <- returns result of default behavior
      you do other stuff (perhaps modify
      default behavior after it occurred)
      set DWLP_MSGRESULT to desired result
   <- return TRUE
   retrieve DWLP_MSGRESULT
<- return it as message result
```

Given this sketch, you should be able to write it up yourself. Here's what I came up with. I call it a Wndproc-Like Dialog:

```
class WLDialogBox
{
public:
  virtual LRESULT WLDlgProc(
            HWND hdlg, UINT uMsg,
            WPARAM wParam, LPARAM lParam)
  {
    return DefDlgProcEx(hdlg, uMsg, wParam, lParam,
                        &m_fRecursing);
  }
  INT_PTR DoModal(HINSTANCE hinst, LPCTSTR pszTemplate,
                  HWND hwndParent)
  {
    m_fRecursing = FALSE;
    return DialogBoxParam(hinst, pszTemplate, hwndParent,
                          s_DlgProc, (LPARAM)this);
  }
private:
  static INT_PTR CALLBACK s_DlgProc(
            HWND hdlg, UINT uMsg,
            WPARAM wParam, LPARAM lParam)
  {
    if (uMsg == WM_INITDIALOG) {
      SetWindowLongPtr(hdlg, DWLP_USER, lParam);
    }
    WLDialogBox *self = (WLDialogBox*)GetWindowLongPtr(
                            hdlg, DWLP_USER);
    if (!self) {
      return FALSE;
    }
    CheckDefDlgRecursion(&self->m_fRecursing);
    return SetDlgMsgResult(hdlg, uMsg,
              self->WLDlgProc(
                hdlg, uMsg, wParam, lParam));
  }
private:
  BOOL m_fRecursing;
};
```

Let’s walk through this class.

The `WLDlgProc` method is virtual because we expect derived classes to do custom actions in their dialog procedure that we invoke from our `s_DlgProc`. The default implementation in the base class uses the `DefDlgProcEx` macro from `windowsx.h` to do the dirty work. That’s right, this technique has been published by Microsoft since 1992. If you look at `DefDlgProcEx`, it sets the recursion flag to TRUE and then calls `DefDlgProc`, which triggers the recursive call.

I could have had a separate `WLDefDlgProc` method which calls `DefDlgProcEx` and have `WLDlgProc` call `WLDefDlgProc`. (In fact, my first version did exactly that.) But I decided not to have a `WLDefDlgProc` to remove the temptation to bypass the base class's `WLDefDlgProc`. Instead, if you want default handling to take place, forward the call to your base class's `WLDefDlgProc`.

The `s_DlgProc` method is the dialog procedure used for all instances of Wndproc-Like dialogs. It initializes itself in the `WM_INITDIALOG` message so future messages can identify which instance of the dialog is handling the message. After short-circuiting messages that arrive before the dialog box has initialized, it uses the `CheckDlgRecursion` macro, also from `windowsx.h`. This macro checks the recursion flag; if set, then it resets the flag and just returns FALSE immediately. This is what stops the recursion. Otherwise, it calls the `WLDlgProc` method (which has probably been overriden in a derived class), then sets the dialog procedure return value and returns.

The `SetDlgMsgResult` macro also comes from `windowsx.h`: It stores the return value into the `DWLP_MSGRESULT` and returns TRUE. Well, unless the message is one of the special exceptions, in which case it returns the value directly. **Note to 64-bit developers**: There is a bug in this macro as currently written. The expression `(BOOL)(result)` should be changed to `(INT_PTR)(result)` so that the upper 32 bits of the return value is not truncated.

The last method is `DoModal`, which initializes the recursion flag and kicks off the dialog box.

Here's a sample program that illustrates the use of this class:

```
class SampleWLDlg : public WLDialogBox
{
  LRESULT WLDlgProc(HWND hdlg, UINT uMsg,
            WPARAM wParam, LPARAM lParam)
  {
    switch (uMsg) {
    HANDLE_MSG(hdlg, WM_COMMAND, OnCommand);
    HANDLE_MSG(hdlg, WM_SETCURSOR, OnSetCursor);
    }
    return __super::WLDlgProc(hdlg, uMsg, wParam, lParam);
  };
  void OnCommand(HWND hdlg, int id,
                 HWND hwndCtl, UINT codeNotify)
  {
    switch (id) {
    case IDCANCEL:
      MessageBox(hdlg, TEXT("Bye"), TEXT("Title"), MB_OK);
      EndDialog(hdlg, 1);
      break;
    }
  }
  BOOL OnSetCursor(HWND hdlg, HWND hwndCursor,
                   UINT codeHitTest, UINT msg)
  {
    if (codeHitTest == HTCAPTION) {
      SetCursor(LoadCursor(NULL, IDC_SIZEALL));
      return TRUE;
    }
    return FORWARD_WM_SETCURSOR(hdlg, hwndCursor,
                  codeHitTest, msg, __super::WLDlgProc);
  }
};
int WINAPI WinMain(HINSTANCE hinst, HINSTANCE hinstPrev,
                   LPSTR lpCmdLine, int nShowCmd)
{
    SampleWLDlg dlg;
    dlg.DoModal(hinst, MAKEINTRESOURCE(1), NULL);
    return 0;
}
1 DIALOGEX DISCARDABLE  0, 0, 200,200
STYLE DS_SHELLFONT | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "sample"
FONT 8, "MS Shell Dlg"
BEGIN
DEFPUSHBUTTON "&Bye",IDCANCEL,"Button",WS_TABSTOP,7,4,50,14
END
```

To illustrate a custom return value, I override the `WM_SETCURSOR` message to display a custom cursor when the mouse is over the caption area. It's not exciting, but it gets the point across.

Observe that in two places, we invoked the default handler by calling `__super::WLDlgProc`. `__super` is a Visual C++ extension that resolves to the base class of your derived class. This is quite handy since it saves the reader the trouble of figure out "So which level in the class hierarchy are we forwarding this call to?" If you wanted to forward a call to your grandparent class, you would use `__super::__super::WLDlgProc`.

If your compiler doesn't support `__super`, you can fake it by adding this line to the definition of `SampleWLDlg`:

```
  typedef WLDialogBox super;
```

and using `super::WLDlgProc` without the underscores. In fact, this is the technique I use because I was doing it before the VC folks added the `__super` keyword and now it's just habit.

Exercise: Does the `m_fRecursing` member really have to be per-instance? Can it be global?

Raymond Chen

Follow