Another different type of dialog procedure

devblogs.microsoft.com/oldnewthing/20031113-00

November 13, 2003

Raymond Chen

The other method of using a window-procedure-like dialog box is to change the rules of the game. Normally, the window procedure for a dialog box is the `DefDlgProc` function, which calls the dialog procedure and then takes action if the dialog procedure indicated that it desired the default action to take place.

The dialog procedure is subservient to `DefDlgProc`, providing advice when requested. The kernel of the idea for this technique is to "turn the tables". Make `DefDlgProc` be the one who gives advice and you be the one that asks for the advice when you want it.

We do this by making the window procedure be our own function which decides whether or not it wants the default action to happen. If so, it calls `DefDlgProc` to do it, after giving the dialog a dummy dialog procedure that always says "Just do the default".

Here's the flow diagram:

```
Message delivered
-> WLWndProc
   -> your WLDlgProc
      decide what to do
      want to do the default action
      -> DefDlgProc
         -> dummy dialog procedure
         <- always returns FALSE
         DefDlgProc does default action
      <- returns result of default behavior
      you do other stuff (perhaps modify
      default behavior after it occurred)
   <- returns result
<- returns result
```

To do this, we need to register a custom dialog class. You always wondered what that was for: Now you know.

```
BOOL
InitApp(void)
{
  WNDCLASS wc;
  wc.style = CS_DBLCLKS | CS_SAVEBITS | CS_BYTEALIGNWINDOW;
  wc.lpfnWndProc = WLWndProc;
  wc.cbClsExtra = 0;
  wc.cbWndExtra = DLGWINDOWEXTRA + sizeof(WLDLGPROC);
  wc.hInstance = g_hinst;
  wc.hIcon = NULL;
  wc.hCursor = LoadCursor(NULL, IDC_ARROW);
  wc.hbrBackground = NULL;
  wc.lpszMenuName = NULL;
  wc.lpszClassName = TEXT("WLDialog");
  if (!RegisterClass(&wc)) return FALSE;
  return TRUE;
}
```

This creates a new window class called “WLDialog” which we will use as our custom dialog class. When you create a custom dialog class, you must set the cbWndExtra to DLGWINDOWEXTRA bytes, plus any additional bytes you wish to use for yourself. We need to store an extra WLDLGPROC, so we add that in.

To use our custom dialog procedure, the dialog template must use the “CLASS” keyword to specify the custom dialog class:

```
1 DIALOGEX DISCARDABLE  0, 0, 200,200
STYLE DS_SHELLFONT | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CLASS "WLDialog"
CAPTION "sample"
FONT 8, "MS Shell Dlg"
BEGIN
    DEFPUSHBUTTON "&Bye", IDCANCEL, 7,4,50,14, WS_TABSTOP
END
```

This is exactly the same as a regular dialog box template, except that there is a “CLASS” entry which specifies that this dialog box should use our new class. Paralleling the `DialogBoxParam` function we have our own:

```
typedef LRESULT (CALLBACK* WLDLGPROC)(HWND, UINT, WPARAM, LPARAM);
struct WLDIALOGINFO {
  WLDLGPROC wldp;
  LPARAM lParam;
};
INT_PTR
WLDialogBoxParam(HINSTANCE hinst, LPCTSTR pszTemplate,
  HWND hwndParent, WLDLGPROC wldp, LPARAM lParam)
{
  WLDIALOGINFO wldi = { wldp, lParam };
  return DialogBoxParam(hinst, pszTemplate,
           hwndParent, WLDlgProc, (LPARAM)&wldi);
}
```

This packages up the WndProc-Like dialog procedure and its reference data so we can recover it in our window procedure:

```
LRESULT CALLBACK
WLWndProc(HWND hdlg, UINT uiMsg, WPARAM wParam, LPARAM lParam)
{
  if (uiMsg == WM_INITDIALOG) {
    WLDIALOGINFO *pwldi = (WLDIALOGINFO*)lParam;
    SetWindowLongPtr(hdlg, DLGWINDOWEXTRA, (LONG_PTR)pwldi->wldp);
    lParam = pwldi->lParam;
  }
  WLDLGPROC wldp = (WLDLGPROC)GetWindowLongPtr(hdlg, DLGWINDOWEXTRA);
  if (wldp) {
    return wldp(hdlg, uiMsg, wParam, lParam);
  } else {
    return DefDlgProc(hdlg, uiMsg, wParam, lParam);
  }
}
```

This is the window procedure for the custom dialog. When the `WM_INITDIALOG` message comes in, we recover the original parameters to `WLDialogBoxParam`. The `WLDLGPROC` we save in the extra bytes we reserved, and the original `LPARAM` becomes the `lParam` that we pass to the `WLDLGPROC`. Then for each message that comes in, we pass the message and its parameters directly to the `WLDLGPROC` and return the value directly. No `DWLP_MSGRESULT` necessary.

The last piece of the puzzle is the dialog procedure we actually hand to the dialog manager:

```
INT_PTR CALLBACK
WLDlgProc(HWND hdlg, UINT uiMsg, WPARAM wParam, LPARAM lParam)
{
  return FALSE;
}
```

All it says is, “Do the default thing.”

Okay so let's write yet another version of our sample program, using this new architecture:

```
LRESULT CALLBACK SampleWLDialogProc(
HWND hdlg, UINT uiMsg, WPARAM wParam, LPARAM lParam)
{
  switch (uiMsg) {
  case WM_INITDIALOG:
    break;
  case WM_COMMAND:
    switch (GET_WM_COMMAND_ID(wParam, lParam)) {
    case IDCANCEL:
      MessageBox(hdlg, TEXT("Bye"), TEXT("Title"), MB_OK);
      EndDialog(hdlg, 1);
      break;
    }
    break;
  case WM_SETCURSOR:
    if (LOWORD(lParam) == HTCAPTION) {
      SetCursor(LoadCursor(NULL, IDC_SIZEALL));
      return TRUE;
    }
    break;
  }
  return DefDlgProc(hdlg, uiMsg, wParam, lParam);
}
int WINAPI WinMain(HINSTANCE hinst, HINSTANCE hinstPrev,
                   LPSTR lpCmdLine, int nShowCmd)
{
  InitApp();
  WLDialogBoxParam(hinst, MAKEINTRESOURCE(1),
            NULL, SampleWLDialogProc, 0);
  return 0;
}
```

In this style of WndProc-Like dialog, we just write our dialog procedure as if it were a window procedure, calling `DefDlgProc()` to perform default behavior. And to get this new behavior, we use `WLDialogBoxParam` instead of `DialogBoxParam`

So now I've developed two quite different ways you can write WndProc-Like dialog procedures. You may not like either one of them, so go ahead and write a third way if you prefer. But at least I hope you learned a little more about how Windows works.

Raymond Chen

Follow