The dialog manager, part 4: The dialog loop

devblogs.microsoft.com/oldnewthing/20050401-00

April 1, 2005

Raymond Chen

The dialog loop is actually quite simple. At its core, it's just

```
while (<dialog still active> &&
       GetMessage(&msg, NULL, 0, 0, 0)) {
 if (!IsDialogMessage(hdlg, &msg)) {
  TranslateMessage(&msg);
  DispatchMessage(&msg);
 }
}
```

If you want something fancier in your dialog loop, you can take the loop above and tinker with it.

But let's start from the beginning. The work happens in DialogBoxIndirectParam. (You should already know by now how to convert all the other DialogBoxXxx functions into DialogBoxIndirectParam.)

```
INT_PTR WINAPI DialogBoxIndirectParam(
    HINSTANCE hinst,
    LPCDLGTEMPLATE lpTemplate, HWND hwndParent,
    DLGPROC lpDlgProc, LPARAM lParam)
{
 /*
  * App hack!  Some people pass GetDesktopWindow()
  * as the owner instead of NULL.  Fix them so the
  * desktop doesn't get disabled!
  */
 if (hwndParent == GetDesktopWindow())
  hwndParent = NULL;
```

That's right, we start with an app hack. The problem of passing GetDesktopWindow() instead of NULL was discussed in an earlier entry. So many people make this mistake that we had to put this app hack into the core OS. It would be pointless to make a shim for it since that would mean that thousands of apps would need to be shimmed.

Since only top-level windows can be owners, we have to take the putative `hwndParent` (which might be a child window) and walk up the window hierarchy until we find a top-level window.

```
if (hwndParent)
 hwndParent = GetAncestor(hwndParent, GA_ROOT);
```

With that second app hack out of the way, we create the dialog.

```
HWND hdlg = CreateDialogIndirectParam(hinst,
              lpTemplate, hwndParent, lpDlgProc,
              lParam);
```

Note: As before, I am going to ignore error checking and various dialog box esoterica because it would just be distracting from the main point of this entry.

Modal windows disable their parent, so do it here.

```
BOOL fWasEnabled = EnableWindow(hwndParent, FALSE);
```

We then fall into the dialog modal loop:

```
MSG msg;
while (<dialog still active> &&
       GetMessage(&msg, NULL, 0, 0)) {
 if (!IsDialogMessage(hdlg, &msg)) {
  TranslateMessage(&msg);
  DispatchMessage(&msg);
 }
}
```

Per the convention on quit messages, we re-post any quit message we may have received so the next outer modal loop can see it.

```
if (msg.message == WM_QUIT) {
 PostQuitMessage((int)msg.wParam);
}
```

(Astute readers may have noticed an uninitialized variable bug: If EndDialog was called during WM_INITDIALOG handling, then msg.message is never set. I decided to ignore this fringe case for expository purposes.)

Now that the dialog is complete, we clean up. Remember to enable the owner before destroying the owned dialog.

```
if (fWasEnabled)
 EnableWindow(hwndParent, TRUE);
DestroyWindow(hdlg);
```

And that's all. Return the result.

```
 return <value passed to EndDialog>;
}
```

Congratulations, you are now an expert on dialog boxes. Tomorrow we'll look at how you can put this new expertise to good use.

Exercise: Find a way to sneak through the two layers of `hwndParent` parameter "repair" and end up with a dialog box whose owner is the desktop window. Explain the dire consequences of this scenario.

Raymond Chen

Follow