The dialog manager, part 6: Subtleties in message loops

devblogs.microsoft.com/oldnewthing/20050405-46

April 5, 2005

Raymond Chen

Last time, I left you with a homework exercise: Find the subtle bug in the interaction between `EndManualModalDialog` and the modal message loop.

The subtlety is that `EndManualModalDialog` sets some flags but does nothing to force the message loop to notice that the flag was actually set. Recall that the `GetMessage` function does not return until a posted message arrives in the queue. If incoming sent messages arrive, they are delivered to the corresponding window procedure, but the `GetMessage` function doesn't return. It just keeps delivering incoming sent messages until a posted message finally arrives.

The bug, therefore, is that when you call `EndManualModalDialog`, it sets the flag that tells the modal message loop to stop running, but doesn't do anything to ensure that the modal message loop will wake up to notice. Nothing happens until a posted message arrives, which causes `GetMessage` to return. The posted message is dispatched and the `while` loop restarted, at which point the code finally notices that the `fEnded` flag is set and breaks out of the modal message loop.

There are a few ways of fixing this problem. The quick solution is to post a meaningless message.

```
void EndManualModalDialog(HWND hdlg, int iResult)
{
 DIALOGSTATE *pds = reinterpret_cast<DIALOGSTATE*>
     (GetWindowLongPtr(hdlg, DWLP_USER));
 if (pds) {
  pds->iResult = iResult;
  pds->fEnded = TRUE;
  PostMessage(hdlg, WM_NULL, 0, 0);
 }
}
```

This will force the `GetMessage` to return, since we made sure there is at least one posted message in the queue waiting to be processed. We chose the `WM_NULL` message because it doesn't do anything. We aren't interested in what the message does, just the fact that there is a message at all.

Next time, a different solution to the same problem.

Raymond Chen

Follow