Modality, part 1: UI-modality vs code-modality

devblogs.microsoft.com/oldnewthing/20050218-00

February 18, 2005

Raymond Chen

From the end-users' point of view, modality occurs when the users are locked into completing a task once it is begun, with the only escape being to cancel the entire operation. Opening a file is an example of a modal operation: Once the "Open" command has been selected, users have no choice but to select a file for opening (or to cancel the operation). While attempting to open a document, the users cannot interact with the existing document (for example, scroll it around to look for some text that would give a clue as to what file to open next).

From a programmer's point of view, modality can be viewed as a function that performs some UI and doesn't return until that UI is complete. In other words, modality is a nested message loop that continues processing messages until some exit condition is reached. In our example above, the modality is inherent in the `GetOpenFileName` function, which does not return until the user selects a filename or cancels the dialog box.

Note that these concepts do not necessarily agree. You can create something that is UI-modal—that is, does not let the user interact with the main window until some other action is complete—while internally coding it as a non-modal function.

Let's code up an example of this behavior, to drive the point home.

As always, start with our scratch program.

```
#include <commdlg.h>
HWND g_hwndFR;
TCHAR g_szFind[80];
FINDREPLACE g_fr = { sizeof(g_fr) };
UINT g_uMsgFindMsgString;
void CreateFindDialogUIModally(HWND hwnd)
{
  if (!g_hwndFR) {
    g_uMsgFindMsgString = RegisterWindowMessage(FINDMSGSTRING);
    if (g_uMsgFindMsgString) {
      g_fr.hwndOwner = hwnd;
      g_fr.hInstance = g_hinst;
      g_fr.lpstrFindWhat = g_szFind;
      g_fr.wFindWhatLen = 80;
      g_hwndFR = FindText(&g_fr);
    }
  }
}
void OnChar(HWND hwnd, TCHAR ch, int cRepeat)
{
  switch (ch) {
  case ' ': CreateFindDialogUIModally(hwnd); break;
  }
}
void OnFindReplace(HWND hwnd, FINDREPLACE *pfr)
{
  if (pfr->Flags & FR_DIALOGTERM) {
      DestroyWindow(g_hwndFR);
      g_hwndFR = NULL;
  }
}
// Add to WndProc
    HANDLE_MSG(hwnd, WM_CHAR, OnChar);
    default:
      if (uiMsg == g_uMsgFindMsgString && g_uMsgFindMsgString) {
        OnFindReplace(hwnd, (FINDREPLACE*)lParam);
      }
      break;
// Edit WinMain
    while (GetMessage(&msg, NULL, 0, 0)) {
        if (g_hwndFR && IsDialogMessage(g_hwndFR, &msg)) {
        } else {
          TranslateMessage(&msg);
          DispatchMessage(&msg);
        }
    }
```

This is an unexciting example of a modeless dialog; in our case, the Find dialog is displayed when you hit the space bar. Observe that you can click back to the main window while the Find dialog is up; that's because the Find dialog is modeless. As is typical for modeless

dialogs, dispatching its messages is handled in the main message loop with a call to the `IsDialogMessage` function.

We can turn this into a UI-modal dialog very simply:

```
void CreateFindDialogUIModally(HWND hwnd)
{
  if (!g_hwndFR) {
    g_uMsgFindMsgString = RegisterWindowMessage(FINDMSGSTRING);
    if (g_uMsgFindMsgString) {
      g_fr.hwndOwner = hwnd;
      g_fr.hInstance = g_hinst;
      g_fr.lpstrFindWhat = g_szFind;
      g_fr.wFindWhatLen = 80;
      g_hwndFR = FindText(&g_fr);
      if (g_hwndFR) {
        EnableWindow(hwnd, FALSE);
      }
    }
  }
}
void OnFindReplace(HWND hwnd, FINDREPLACE *pfr)
{
  if (pfr->Flags & FR_DIALOGTERM) {
      EnableWindow(hwnd, TRUE);
      DestroyWindow(g_hwndFR);
      g_hwndFR = NULL;
  }
}
```

Notice that we carefully observed the rules for enabling and disabling windows.

When you run this modified program, everything seems the same except that the Find dialog is now modal. You can't interact with the main window until you close the Find dialog. The Find dialog is modal in the UI sense. However, the code is structured in the non-modal manner. There is no dialog loop; the main window loop dispatches dialog messages as necessary.

One typically does not design one's modal UI in this manner because it makes the code harder to structure. Observe, for example, that the code to manage the dialog box is scattered about and the management of the dialog needs to be handled as a state machine since each phase returns back to the main message loop.

Raymond Chen

Follow