Those who do not understand the dialog manager are doomed to reimplement it, badly

devblogs.microsoft.com/oldnewthing/20070627-00

June 27, 2007

Raymond Chen

A customer wanted to alter the behavior of a multi-line edit control so that it did not treat a press of the Tab key as a request to insert a tab character but rather treated it as a normal dialog navigation key. The approach the customer took was to subclass the edit control and intercept the Tab key:

```
LRESULT CALLBACK SubclassWndProc(
    HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
  switch (uMsg) {
  case WM_KEYDOWN:
    if (wParam == VK_TAB) {
      // Pressed the TAB key - tab to next control
      SetFocus(GetNextDlgTabItem(
                    GetParent(hwnd), hwnd, FALSE));
      return 0; // message handled
    }
  }
  return CallWindowProc(...);
}
```

There are many things wrong with this approach. You can spend quite a lot of time nitpicking the little details, how this code fails to set focus in a dialog box properly, how it fails to take nested dialogs into account, how it fails to handle the Shift+Tab navigation key, how it blatantly assumes that the control is part of a dialog box in the first place! But all of these little details are missing the big picture: Instead of fighting against the dialog manager and reimplementing all the parts we want to keep and ignoring the parts we want to skip, we should be working *with* the dialog manager and expressing our intentions in the manner the dialog manager expects.

It's the difference between ordering a hamburger without pickles and ordering a hamburger with pickles, and then carefully picking the pickles off the burger when you get it.

In this case, we want to prevent the edit control from saying "Give me the Tab key." We saw last time that this is done either by (1) setting the `DLGC_WANTTAB` dialog code or by (2) responding with `DLGC_WANTMESSAGE` when given a Tab key message. Therefore, to tell the dialog manager to not to treat the tab key specially, just turn off those two behaviors.

```
LRESULT CALLBACK SubclassWndProc(
    HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
  LRESULT lres;
  switch (uMsg) {
  case WM_GETDLGCODE:
    lres = CallWindowProc(...);
    lres &= ~DLGC_WANTTAB;
    if (lParam &&
        ((MSG *)lParam)->message == WM_KEYDOWN &&
        ((MSG *)lParam)->wParam == VK_TAB) {
      lres &= ~DLGC_WANTMESSAGE;
    }
    return lres;
  }
  return CallWindowProc(...);
}
```

After asking the original control what behavior it thinks it wants, we turn off the `DLGC_WANTTAB` flag; this takes care of part (1). Next, we check whether the message is a press of the Tab key. If so, then we turn the `DLGC_WANTMESSAGE` flag off; this takes care of part (2).

This is certainly less code than would need to have been written to address all of the little concerns noted earlier, and it does it by completely sidestepping the task of trying to emulate the dialog manager and instead just cooperating with the dialog manager to get the behavior you want. This principle of "If you know how a system is meant to work, you can work with it rather than against it, and everybody will be happier for it" is something I've been trying to convey through this web site and my book. Knowing how something was intended to be used allows you to be a more effective programmer.

Exercise: Why didn't we just write this instead?

```
LRESULT CALLBACK SubclassWndProc(
    HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
  LRESULT lres;
  switch (uMsg) {
  case WM_GETDLGCODE:
    lres = CallWindowProc(...);
    lres &= ~DLGC_WANTTAB;
    if (wParam == VK_TAB) {
      lres &= ~DLGC_WANTMESSAGE;
    }
    return lres;
  }
  return CallWindowProc(...);
}
```

Raymond Chen

Follow