How can I create a non-circular tab order, or some other type of custom ordering in my Win32 dialog?

devblogs.microsoft.com/oldnewthing/20201231-00

December 31, 2020

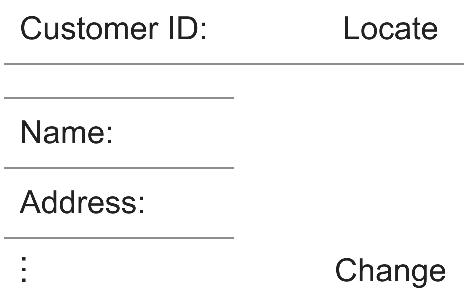
Raymond Chen

Normally, the tab order in a dialog follows a fixed sequence: Hitting the `Tab` key moves forward through the sequence, and hitting `Shift`+`Tab` moves backward through the sequence, wrapping around when the beginning or end of the sequence is reached. In other words, what you have is a circle.

The order is based on the order in which the controls are given in the dialog template, which need not match the physical layout of the controls. In other words, if you list a control near the bottom of the dialog ahead of a control near the top, then hitting the `Tab` key will move from the bottom control to the top control. This can be handy if you want the tab order to move vertically through columns, say.

But sometimes a circular order isn't good enough.

Say you have a dialog box that looks in part like this:



The idea is that the user enters the customer ID into the edit box, and then clicks the *Locate* button. This looks up the customer record, and the user can then use other buttons on the dialog to view details of the customer or make changes.

Based on end-user feedback, you come to the conclusion that it would be better if tabbing backward from the *Change* button went straight to the *Customer ID* field, rather than to the *Locate* button. After all, there's no point clicking the *Locate* button without first making a change to the customer ID.

You can do this by overriding the tab behavior for the *Change* button.

```
INT_PTR CALLBACK CustomerDlgProc(
    HWND hdlg, UINT message, WPARAM wParam, LPARAM lParam)
{
  switch (message) {
  case WM_INITDIALOG:
    SetWindowSubclass(GetDlgItem(hDlg, IDC_CHANGENAME),
                      TabBackwardSubclassProc, 0, 0);
    ... other initialization ...
    return TRUE;

  case ...
  }
  return FALSE;
}

INT_PTR CALLBACK TabBackwardSubclassProc(
    HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam,
    UINT_PTR subclassId, DWORD_PTR)
{
  switch (message) {
  case WM_NCDESTROY:
    RemoveWindowSubclass(hwnd, TabBackwardSubclassProc,
                         subclassId);
    break;

  case WM_GETDLGCODE:
    return DefSubclassProc(hwnd, message, wParam, lParam) |
           DLGC_WANTTAB;

  case WM_KEYDOWN:
    if (wParam == VK_TAB) {
      HWND hdlg = GetParent(hwnd);
      if (GetKeyState(VK_SHIFT) < 0) {
        // Tabbing backward - go to the Customer ID.
        HWND tabDestination = GetDlgItem(hdlg,
                                         IDC_CUSTOMERID);
        SendMessage(hdlg, WM_NEXTDLGCTL,
                    (WPARAM)tabDestination, TRUE);
      } else {
        // Do the normal tabbing thing.
        SendMessage(hdlg, WM_NEXTDLGCTL, FALSE, FALSE);
      }
      return 0;
    }
    break;

    case WM_CHAR:
      if (wParam == VK_TAB) return 0;
      break;
    }
    return DefSubclassProc(hwnd, message, wParam, lParam);
}
```

During dialog box initialization, we subclass the control for which we want a custom tab destination. In our case, it's the *Change* button.

In the subclass procedure, there is the usual boilerplate about removing the subclass when the window is destroyed. But the interesting part starts with the `WM_ GETDLGCODE` message.

As I noted some time ago, the `WM_GETDLGCODE` message lets you influence the behavior of the dialog manager. We handle this message by taking the behavior requested by the original control, and also saying that we want to customize the behavior of the `Tab` key.

Doing so allows the `VK_TAB` key to flow into the `WM_ KEYDOWN`, `WM_ KEYUP`, and `WM_ CHAR` messages.

When a key goes down, we trigger our custom navigation when the `Tab` key is pressed. If the `Shift` key is also pressed, then we use the `WM_NEXTDLGCTL` message to move focus to an explicit control: Passing an `lParam` of `TRUE` means that we are specifying the window to go to, and we give it the *Customer ID* control.

If the `Shift` key is not pressed, then we pass an `lParam` of `FALSE`, meaning "Do default tab navigation." Passing `FALSE` as the `wParam` means that we should go to the default *next* control. (Passing `TRUE` would request going to the default *previous* control.)

The last bit of cleanliness is that we need to grab the `WM_ CHAR` message and swallow the `Tab` character, so that the control itself won't try to respond to it, say, by inserting a tab into the edit control.

And there you have it. We customized tabbing backward from the *Change* button in a way that resulted in a tab order that isn't circular.

Raymond Chen

Follow