When the default pushbutton is invoked, the invoke goes to the top-level dialog

devblogs.microsoft.com/oldnewthing/20120621-00

June 21, 2012

Raymond Chen

One quirk of nested dialogs lies in what happens when the user presses Enter to invoke the default pushbutton: The resulting `WM_COMMAND` message *goes to the top-level dialog*, even if the default pushbutton belongs to a sub-dialog.

Why doesn't it send the `WM_COMMAND` to the parent of the default pushbutton? I mean, the dialog manager knows the handle of the button, so it can send the message to the button's parent, right?

Well, the dialog manager knows the handle of *a* button. But not necessarily *the* button. Recall that if focus is not on a pushbutton, then the dialog manager sets the default pushbutton based on the control ID returned by the `DM_GETDEFID` message, and it does this by just searching the dialog for a control with that ID. If you have two controls with the same ID, it picks one of them arbitrarily. So far so bad.

It's like having two John Smiths living in your house, one in the second bedroom and one living in the guest room. The post office is very strict and won't let you write "John Smith, Second Bedroom, 1 Main Street" and "John Smith, Guest Room, 1 Main Street." All you're allowed to write is a name and an address. Therefore, all the mail addressed to "John Smith, 1 Main Street" ends up in a single mailbox labeled "1 Main Street" and now you have to figure out who gets each piece of mail.

Okay, so we saw that when converting an ID to a window, and there are multiple windows with the same ID, the dialog manager will just pick one arbitrarily. And if it picks the wrong one, it would have sent the `WM_COMMAND` to the wrong dialog procedure entirely! At least by sending it to the top-level dialog, it says, "Dude, I think it's this window but I'm not sure, so if you have some really clever way of telling which is which, you can try to sort it out." And now that the `WM_COMMAND` *sometimes* goes to the top-level dialog, you're pretty much stuck having it *always* go to the top-level dialog for consistency. It's better to be consistently wrong in a predictable manner (so people can work around it reliably) than to be mostly-right and occasionally-completely-wrong.

Third rationale: Because you're asking for code to be written to handle a case that people shouldn't have gotten into in the first place. (Namely, duplicate control IDs.)

Whatever the reason, it's something you need to be on the lookout for. If you did everything right and avoided control ID duplication, then the workaround in your `WM_COMMAND` handler is straightforward:

```
void OnCommand(HWND hwnd, int id, HWND hwndCtl, UINT codeNotify)
{
    if (hwndCtl != nullptr)
    {
        HWND hwndCtlParent = GetParent(hwndCtl);
        if (hwndCtlParent != nullptr &&
            hwndCtlParent != hwnd &&
            IsChild(hwnd, hwndCtlParent))
        {
           FORWARD_WM_COMMAND(hwndCtlParent, id,
                              hwndCtl, codeNotify, SendMessage);
           return;
        }
    }
    ... the message was for me after all, so let's handle it...
    switch (id)
    {
    ...
    }
}
```

When we get the `WM_COMMAND` message, we first check that it really came from one of our direct children. If not, then we forward the message on to the control's actual parent. (The window that should have gotten the message in the first place in an ideal world.)

Exercise: Under what circumstances can the above workaround fail? (Not counting the scenario we've spent the past three days discussing.)

Anyway, back to the question from last time: How does the property sheet manager deal with multiple property sheets pages having conflicting control IDs? In addition to what we previously discussed, another mitigating factor is that the property sheet manager keeps only one child dialog visible at a time. This takes the hidden child dialogs out of the running for most dialog-related activities, such as dialog navigation, since invisible controls cannot be targets of dialog navigation. Furthermore, hidden child dialogs are skipped when searching for keyboard accelerators, thereby avoiding the problem of hidden accelerators. So as long as the property sheet manager makes sure that focus doesn't stay on a hidden control after a page change, there shouldn't be any notifications coming from a hidden child dialog. The only conflicts it needs to worry about are conflicts between the page and the frame.

Raymond Chen

Follow