Why do DLGC_WANTALLKEYS and DLGC_WANTMESSAGE have the same value?

devblogs.microsoft.com/oldnewthing/20070626-00

June 26, 2007

Raymond Chen

From a purely theoretical point of view, there is only one "want" code you really need: `DLGC_WANTMESSAGE`. All the others are just conveniences. For example, returning `DLGC_WANTARROWS` means "I want this message if it is an arrow key; otherwise, I don't care." It lets you write

```
case WM_GETDLGCODE:
 return DLGC_WANTARROWS;
```

instead of the more cumbersome (but equivalent)

```
case WM_GETDLGCODE:
 if (lParam &&
     ((MSG*)lParam)->message == WM_KEYDOWN &&
     (wParam == VK_LEFT || wParam == VK_RIGHT ||
      wParam == VK_UP || wParam == VK_DOWN)) {
   return DLGC_WANTMESSAGE;
 }
 return 0;
```

Similarly, `DLGC_WANTTAB` is equivalent to returning `DLGC_WANTMESSAGE` if the message is a press of the tab key, and `DLGC_WANTCHARS` is equivalent to returning `DLGC_WANTMESSAGE` if the message is `WM_CHAR`.

And that leaves `DLGC_WANTALLKEYS`, which is just another name for `DLGC_WANTMESSAGE`:

```
#define DLGC_WANTALLKEYS    0x0004
#define DLGC_WANTMESSAGE    0x0004
```

They mean the same thing but look at the situation through different perspectives. The `DLGC_WANTMESSAGE` value is more readable if you return it as part of some larger decision-making process, like we did with our mimicry of `DLGC_WANTTAB`: You do a bunch of tests and then when you decide, "I guess I want this one," you return `DLGC_WANTMESSAGE`. On the other hand, the `DLGC_WANTALLKEYS` value is more readable if you are just returning it unconditionally. "I want all keys, no matter what it is."

It's like when you're at the grocery store, and the bagger asks you, "Would you like me to help you carry your watermelon to your car?" You can say "Yes" or "Always"; the result is the same. The only difference is one of expectation: If you expect to meet the same bagger in the future, and the bagger remembers, then "Always" implies "You don't need to ask me again." The dialog manager, on the other hand, doesn't have that good of a memory, and in fact, if you think about it, you *don't want it to have a good memory*.

Suppose the dialog manager remembered whether you said "Always" and stopped asking you in the future.⊶ It sends the `WM_GETDLGCODE` message to a control, the control responds `DLGC_WANTALLKEYS`, and then later, you subclass the control and change the dialog code. Oops, that doesn't work because the dialog manager "remembered" the control's previous answer and doesn't ask any more.⊷ Naturally, you expect subclassing to work, so the dialog manager asks each time.

Continuing the analogy, if you want the bagger to help you if it is raining but not on a dry day, you can either look out the window, confirm that it is not raining, and say, "Yes", or you can just say, "Yes, if it's raining," regardless of the weather, and let the bagger make the call. (Of course, the analogy breaks down because the bagger might have a different assessment from you as to whether or not it is raining. The decisions you leave up to the dialog manager, such as whether a key is the tab key or not, are much less ambiguous.)

Next time, we'll look at a dialog manager problem, and the information you learned today may come in handy in solving it.

Nitpicker's corner

⊶Beginning of counterfactual discussion.*

⊷End of counterfactual discussion.*

*Warning: Comments complaining about my choice of footnote symbol will be misrepresented and ridiculed.

Raymond Chen

Follow