Use WM_WINDOWPOSCHANGING to intercept window state changes

devblogs.microsoft.com/oldnewthing/20080116-00

January 16, 2008

Raymond Chen

The `WM_WINDOWPOSCHANGING` message is sent early in the window state changing process, unlike `WM_WINDOWPOSCHANGED`, which tells you about what already happened. A crucial difference (aside from the timing) is that you can influence the state change by handling the `WM_WINDOWPOSCHANGING` message and modifying the `WINDOWPOS` structure.

Here's an example that prevents the window from being resized.

```
BOOL OnWindowPosChanging(HWND hwnd, WINDOWPOS *pwp)
{
    pwp->flags |= SWP_NOSIZE;
    /* Continue with default handling */
    return FORWARD_WM_WINDOWPOSCHANGING(hwnd, pwp, DefWindowProc);
}
HANDLE_MSG(hwnd, WM_WINDOWPOSCHANGING, OnWindowPosChanging);
```

Before the `WM_WINDOWPOSCHANGING` message was invented, programs had to enforce window size constraints inside their `WM_SIZE` and `WM_MOVE` handlers, but since those messages are sent *after* the change is complete, the result was flicker as the window changed to one size, then the `WM_SIZE` handler resized it to a better size. Intercepting the window size change in `WM_WINDOWPOSCHANGING` allows you to enforce constraints before the sizing happens, thereby avoiding flicker.

The `WM_WINDOWPOSCHANGING` and `WM_WINDOWPOSCHANGED` pair of messages is just one example of the more general `*CHANGING` / `*CHANGED` pattern. (Other examples are `WM_STYLECHANGING` / `WM_STYLECHANGED` and `LVN_ITEMCHANGING` / `LVN_ITEMCHANGED`.) The `*CHANGING` half is sent before the change takes place, and as a general rule, you can change the parameters of the notification to enforce some type of constraint. After you return from the `*CHANGING` notification, the actual change takes place, and then you receive a `*CHANGED` to indicate that the change is complete.

Raymond Chen

Follow