What's with this MSH_MOUSEWHEEL message?

devblogs.microsoft.com/oldnewthing/20080806-00

August 6, 2008

Raymond Chen

The hardware folks had this mouse wheel thing they were making, and they needed a way to get applications to support the mouse. Now, one way of doing this was to say, "Well, we'll start selling this wheel mouse, but no applications can use it until the next version of Windows is released, one that supports the wheel." Of course, that would have meant waiting until Windows NT 4 came out, and who know when that would be. Plus it meant that people would have to upgrade Windows in order to take advantage of their fancy new mouse. As you can imagine, they weren't too pleased with the "wait a few years" plan.

In the interim, they proposed a stopgap mechanism for applications to respond to the mouse wheel. Enter the `zmouse.h` header file and its `MSH_MOUSEWHEEL` registered message. When you installed the wheel mouse driver, it listened for wheel events from the hardware and posted this new message when the mouse wheel turned, and applications could just respond to either the `WM_MOUSEWHEEL` message (if running on a version of Windows that supported the message) or the `MSH_MOUSEWHEEL` message (if running on an older version of Windows that didn't). Unfortunately, the two messages behave differently, so it's not a simple matter of writing

```
if (uMsg == WM_MOUSEWHEEL || uMsg == g_msgWheel) {
 … do wheel stuff …
}
```

(These next few paragraphs summarize what is already spelled out in MSDN; you can skip them if you already know how the messages work.)

First, let's look at `WM_MOUSEWHEEL`. This message is delivered to the window that has focus (in the `SetFocus` sense). If the window procedure doesn't handle the message and just passes it through to the `DefWindowProc` function, then the `DefWindowProc` function forward the message to the window's parent. In this way, the `WM_MOUSEWHEEL` message automatically "bubbles outward" from the focus window up the parent chain until somebody finally handles the message (or it goes all the way to the top without being handled at all).

On the other hand, the `MSH_MOUSEWHEEL` message works from the outside in. It is delivered to the foreground window (in the `SetForegroundWindow` sense). If the window procedure doesn't want to handle the message, it can forward the message to child windows of its choice, until one of them returns `TRUE` to indicate that the message was handled, or until no further candidates exist.

I'll summarize these differences in a table, since people seem to like tables so much.

	WM_MOUSEWHEEL	**MSH_MOUSEWHEEL**
Propagation direction	Inside-out	Outside-in
Propagation mechanism	DefWindowProc	SendMessage
Handling	Automatic	Manual: Application checks return value from child to determine what to do next
Return value if processed	Zero	TRUE
Return value if not processed	DefWindowProc	FALSE

Notice that `WM_MOUSEWHEEL` is much simpler, and the inside-out propagation mechanism retains the spirit of other messages such as `WM_CONTEXTMENU` and `WM_SETCURSOR`. Why can't `MSH_MOUSEWHEEL` do it the same way?

Well, first of all, `MSH_MOUSEWHEEL` doesn't have the luxury of being able to modify the `DefWindowProc` function. After all, that's the whole point of introducing the message in the first place, because we're trying to add wheel support to an older operating system that predated mouse wheels. Put in other words, if we could modify `DefWindowProc` to handle the `MSH_MOUSEWHEEL` message, then we wouldn't have needed the `MSH_MOUSEWHEEL` message to begin with; we would've just modified `DefWindowProc` to handle the `WM_MOUSEWHEEL` message.

The argument in the previous paragraph is a frustratingly common one. Given a problem X and a workaround Y, somebody will ask, "Why didn't you use method Z?" If you look at method Z, though, you'll see that it suffers from the exact same problem X.

Here's a real-world example of the "confused workaround":

"Since the I-90 bridge is closed, I can't take the 550 bus to get from Bellevue to Safeco Field. Instead, I'm going to take the 230 to Redmond, and then change to the 545."

— Well, that's silly. Why not take the 245 to Eastgate, and then change to the 554? It's a lot faster.

"Um, the 554 uses the I-90 bridge, too."

Okay, so you can't change `DefWindowProc`, but why not at least propagate the `MSH_MOUSEWHEEL` from the inside out instead of from the outside in?

Starting with the focus window assumes you can even find out what the focus window is, but if you had paid attention to the *Five Things Every Win32 Programmer Should Know*, you would have known that each thread has its own focus window. (Not nitpickily true, but true enough.) Consequently, when the helper program that injects `MSH_MOUSEWHEEL` messages calls `GetFocus`, it just gets its own focus window, not the focus window of the thread that controls the foreground window. (Remember, we're talking 1996, long before the `GetGUIThreadInfo` function was invented. History buffs can find out more from Meet The Inventor of the Mouse Wheel.) Since inside-out was off the table, that pretty much forced outside-in.

Now that you know how mouse wheel messages work, you can explain the behavior this customer is seeing:

> I'm seeing the `WM_MOUSEWHEEL` message being delivered to the wrong child window. I have a parent window with two children. Even though I move the mouse pointer over child 1, the `WM_MOUSEWHEEL` goes to child 2.

Raymond Chen

Follow