Untangling the confusingly-named WM_UPDATEUISTATE and WM_CHANGEUISTATE messages

devblogs.microsoft.com/oldnewthing/20130516-00

May 16, 2013

Raymond Chen

I always get confused by the `WM_UPDATEUISTATE` and `WM_CHANGEUISTATE` messages, and I have to go figure them out each time I need to mess with them. So this time, I'm going to write it down so I don't forget. Because the act of writing it down helps me to remember.

It's like in school, where the teacher says, "This is a closed-book, closed-notes exam, but you are allowed to bring one piece of standard 8½″×11″ paper with you, on which you can write anything you like. No funny business." You work really hard to create the ultimate sheet of paper to bring to the exam, and then it turns out that during the exam, you barely refer to it at all. Because the act of deciding what to put on the cheat sheet made you remember the material.

Part of the problem with the messages `WM_UPDATEUISTATE` and `WM_CHANGEUISTATE` is their confusing names, because to most people *update* and *change* are basically the same concept. The difference is the direction the message travels. Before we look at that, let's look at the mysterious `WPARAM`.

The `WPARAM` specifies what action you want to perform (initialize, set, or clear) and the target of the action (focus, accelerators, or both).

Action	Meaning
`UIS_SET`	Set the flag (hide the indicator).
`UIS_CLEAR`	Clear the flag (show the indicator).
`UIS_INITIALIZE`	Set or clear the flag based on whether the last input event was mouse (set) or keyboard (clear).

Setting a flag hides the corresponding indicator. For example, if you have a `UIS_SET` for `UISF_HIDEFOCUS`, that means that you want to hide focus indicators.

Clearing a flag shows the corresponding indicator. For example, if you have a `UIS_CLEAR` for `UISF_HIDEFOCUS`, that means that you want to show focus indicators.

Yes, it's a bit of a double-negative situation.

Each window has its own internal state that remembers which indicators have been hidden for that window. You can query this state by sending the window a `WM_QUERYUISTATE` message.

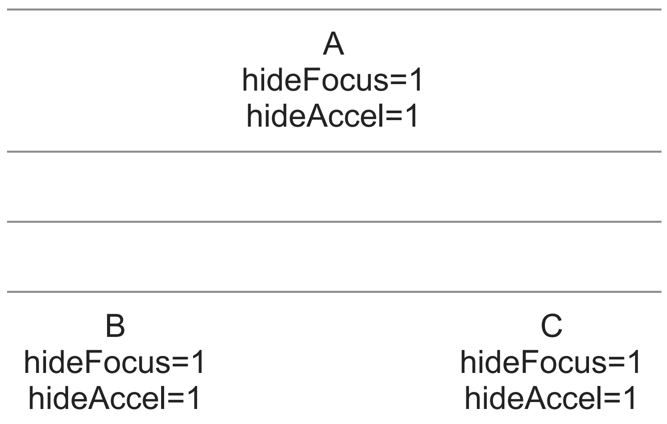
The `WM_UPDATEUISTATE` message travels down the tree: When a window receives the `WM_UPDATEUISTATE` message, it updates its state according to the `WPARAM` and then forwards the message to its children. Therefore, if you want to change the state for an entire window tree, you can send the `WM_UPDATEUISTATE` message to the top-level window, and the message will be delivered to that window and all its children.

It's called *update* because it says, "Okay, listen up everybody, this is what we're going to do."

The `WM_CHANGEUISTATE` message is more like a change *request*. It travels up the tree: When a window receives the message, it sees if the state being requested matches the window's current state. If so, then processing stops since there is nothing to change. Otherwise, the window forwards the message to its parent. The idea here is to push the change request up the tree until it finds the top-level window.

If a top-level window receives a `WM_CHANGEUISTATE` message for a state change that actually changes something, it turns around and sends itself a `WM_UPDATEUISTATE` message, which as we saw before, tells the entire window tree to set its indicator state to the value specified.

Okay, let's draw a picture. Suppose we have a top-level window with two children, and suppose that everybody starts out with all indicators hidden.



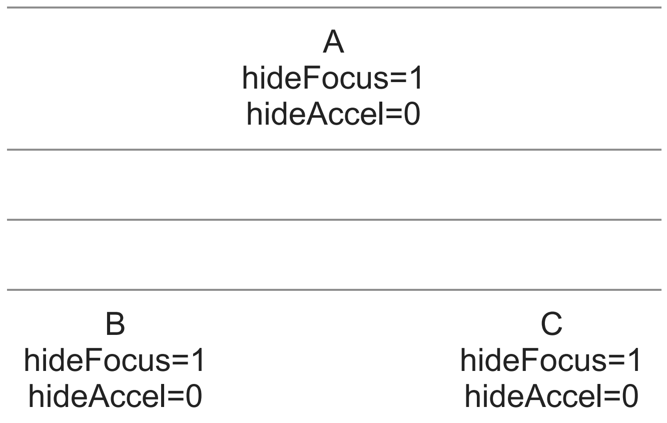
Window B decides that it wants to show accelerators, say because the user tapped the `Alt` key. It sends itself a `WM_CHANGEUISTATE` message with a `wParam` of `MAKEWPARAM(UIS_CLEAR, UISF_HIDEACCEL)`.

The `WM_CHANGEUISTATE` message handler for Window B sees that the `UISF_HIDEACCEL` flag is set, so the *clear* action is meaningful. It forwards the request to its parent, Window A.

The `WM_CHANGEUISTATE` message handler for Window A also sees that the `UISF_HIDEACCEL` flag is set, so the *clear* action is meaningful. Since it has no parent, Window A converts the `WM_CHANGEUISTATE` message to a `WM_UPDATEUISTATE` message and sends it to itself.

The `WM_UPDATEUISTATE` message handler for Window A sees that it is being told to clear the `UISF_HIDEACCEL` flag, so it clears the flag and then forwards the mesage to both its children.

Each of the child windows B and C receive the `WM_UPDATEUISTATE` message and see that they are also being told to clear the `UISF_HIDEACCEL` flag, so they do so. Those windows have no children of their own, so message processing stops. By this mechanism, Window B has managed to convince all the other windows in the hierarchy to clear the `UISF_HIDEACCEL` flag.



Now, suppose that Window C also decides to clear the accelerator indicator. It does the same thing as Window B and sends itself a `WM_CHANGEUISTATE` message with a `wParam` of `MAKEWPARAM(UIS_CLEAR, UISF_HIDEACCEL)`. This time, the `WM_CHANGEUISTATE` message handler for Window C sees that the `UISF_HIDEACCEL` flag is already clear, so the *clear* action is redundant. Message processing stops.

These two examples show the flow of the UI state change messages. When somebody wants to suggest a change to the UI state, they send themselves a `WM_CHANGEUISTATE` message with a description of what they want to change. The above algorithm then kicks in to decide whether the change is meaningful, and if so, it notifies all the other windows in the hierarchy about the new state.

Next time, we'll look at how this whole indicator state thing gets off the ground.

Raymond Chen

Follow