Getting a parent and child window to have the same UI states

devblogs.microsoft.com/oldnewthing/20171124-00

November 24, 2017

Raymond Chen

Last time, we created a program that moved a child window between two parents in such a way that the child and parent ended up with different UI states. We saw the weird things that result from this mismatch, which is why the documentation told us, "you should synchronize the UISTATE of both windows." So let's try it.

Take the program we had from last time and add the following:

```
UINT GetWindowUIState(HWND hwnd)
{
 return LOWORD(SendMessage(hwnd, WM_QUERYUISTATE, 0, 0));
}

LRESULT RootWindow::HandleMessage(
                          UINT uMsg, WPARAM wParam, LPARAM lParam)
{
...
  case WM_LBUTTONDOWN:
   if (GetParent(g_hwndPotato) != m_hwnd) {
    SetParent(g_hwndPotato, m_hwnd);

    // Synchronize the potato's UI state with its new parent.
    auto parentUIState = GetWindowUIState(m_hwnd);
    ResetUIStateForTree(g_hwndPotato, parentUIState);
   }
   break;

...
}
```

The first thing we do is introduce a helper function that gives a pretty name to the `WM_ QUERYUISTATE` message. Since UI states are a 16-bit value, we keep only the least-significant word from the message result.

Meanwhile, in our window procedure, after we reparent the Potato button, we call an as-yet-unimplemented function which tries to reset the UI state for a window tree to a specified value.

Aside: Coming up with a name for that function was a bit of a struggle. The natural name would be `SetUIStateForTree`, but the `WM_UPDATEUISTATE` and `WM_CHANGEUISTATE` messages already use the word "set" to mean "turn on some bits (but don't turn any bits off)". If I had called it `SetUIStateForTree`, then it might have been interpreted to mean that the bits in the `parentUIState` are or'd into the destination UI state rather than copied.

Okay, so let's try to write the `ResetUIStateForTree` function. The only interesting operations available are `UIS_SET` and `UIS_CLEAR`. One of the turns bits on and the other turns them off.

My first attempt at resetting the UI state went like this:

```
// Code in italics is wrong
void ResetUIStateForTree(HWND hwnd, UINT desiredState)
{
 SendMessage(hwnd, WM_UPDATEUISTATE,
  MAKEWPARAM(UIS_SET, desiredState), 0);
 SendMessage(hwnd, WM_UPDATEUISTATE,
  MAKEWPARAM(UIS_CLEAR, ~desiredState), 0);
}
```

We set all the bits that are set and clear all the bits that are clear. We use the `WM_UPDATEUISTATE` message because that is the one that says "Start with the specified window, apply the changes to it, and then propagate the changes to that window's children." If we had used the `WM_CHANGEUISTATE` message, then the request would have propagated out to the root window, and the root window would have done nothing because we are telling the root window to set bits in its UI state that are already set (and clear bits that are already clear).

Some playing around with this version of the function showed that setting the bits that are set was working fine, but clearing the bits that are clear was not.

Reason: Parameter validation.

There are only three bits currently defined for the UI state, so `~uiState` ends up passing a bunch of bits that are invalid, so the call fails.

Okay, so we need to make sure not to try to set or clear bits that are not defined. But I don't want to hard-code the set of valid bits, because a future version of Windows might add a new UI state bit, and I want to be able to reset those too.

This is what I came up with:

```
void ResetUIStateForTree(HWND hwnd, UINT desiredState)
{
 auto currentState = GetWindowUIState(hwnd);
 auto missingState = desiredState & ~currentState;
 if (missingState) {
  SendMessage(hwnd, WM_UPDATEUISTATE,
   MAKEWPARAM(UIS_SET, missingState), 0);
 }
 auto extraState = currentState & ~desiredState;
 if (extraState) {
  SendMessage(hwnd, WM_UPDATEUISTATE,
   MAKEWPARAM(UIS_CLEAR, extraState), 0);
 }
}
```

We take the current state and compare it to the desired state. If there are any bits set in the desired state that aren't set in the current state, then set them. If there are any bits clear in the desired state that aren't clear in the current state, then clear them.

Sure, this sounds really simple, but it took us a few tries to get there.

Restricting the operation to bits that need to be set or cleared ensures that we operate only on bits that the operating system has defined. (If they weren't defined, then the operating system wouldn't have given them to us when we performed the query.)

After making these changes to the program from last time, run it again and play around with setting the keyboard indicator states and moving the Potato window from one window to another. This time, whenever the Potato window moves to a new window, we make its UI state match that of its new parent.

That's quite a lot of complexity packed into the instructions "you should synchronize the UISTATE of both windows."

Exercise: Suppose your UI policy is that the window frame should change its UI state to match the potato it just stole. How would you change our sample program to implement that policy?

Raymond Chen

Follow