WM_NCHITTEST is for hit-testing, and hit-testing can happen for reasons other than the mouse being over your window

devblogs.microsoft.com/oldnewthing/20110218-00

February 18, 2011

Raymond Chen

The `WM_NCHITTEST` message is sent to your window in order determine what part of the window corresponds to a particular point. The most common reason for this is that the mouse is over your window.

- The default `WM_SETCURSOR` handler uses the result of `WM_NCHITTEST` to figure out what type of cursor to show. for example, if you return `HTLEFT`, then `DefWindowProc` will show the `IDC_SIZEWE` cursor.
- If the user clicks the mouse, the default `WM_NCLBUTTONDOWN` handler uses the result of `WM_NCHITTEST` to figure out where on the window you clicked. For example, if you return `HTCLOSE`, then it will act as if the user clicked on the Close button.

Although `WM_NCHITTEST` triggers most often for mouse activity, that is not the only reason why somebody might want to ask, "What part of the window does this point correspond to?"

- The `WindowFromPoint` function uses `WM_NCHITTEST` in its quest to figure out which window is under the point you passed in. If you return `HTTRANSPARENT`, then it will skip your window and keep looking.
- Drag/drop operations use the result of `WM_NCHITTEST` to figure out what part of the window you are dragging over.
- Accessibility tools use the result of `WM_NCHITTEST` to help the user understand what's on the screen.
- Anybody can use the result of `WM_NCHITTEST` to see how your window is laid out. We used it a few years ago to detect a right-click on the caption button.

Consider a program that wants to beep when the mouse is over the Close button. This is an artificial example, but you can use your imagination to come up with more realistic ones, like showing a custom mouseover animation or displaying a balloon tip if the document is unsaved. I chose beeping because it requires less code; otherwise, all the details of its implementation would distract from the point of the example.

Start with the scratch program and make the following changes:

```
BOOL g_fInCloseButton = FALSE;
void EnterCloseButton(HWND hwnd)
{
 if (g_fInCloseButton) return;
 g_fInCloseButton = TRUE;
 MessageBeep(-1); // obviously something more interesting goes here
 TRACKMOUSEEVENT tme = { sizeof(tme), TME_NONCLIENT | TME_LEAVE, hwnd };
 TrackMouseEvent(&tme);
}
void LeaveCloseButton(HWND hwnd)
{
 if (g_fInCloseButton) {
  // stop animation, remove balloon, etc.
  g_fInCloseButton = FALSE;
 }
}
// This code is wrong - see text
UINT OnNcHitTest(HWND hwnd, int x, int y)
{
 UINT ht = FORWARD_WM_NCHITTEST(hwnd, x, y, DefWindowProc);
 if (ht == HTCLOSE) {
  EnterCloseButton(hwnd);
 } else {
  LeaveCloseButton(hwnd);
 }
 return ht;
}
HANDLE_MSG(hwnd, WM_NCHITTEST, OnNcHitTest);
case WM_NCMOUSELEAVE:
 LeaveCloseButton(hwnd);
 break;
```

We keep track of whether or not the mouse is in the close button so that we don't double-start the animation or double-cancel it. (For us, this keeps us from beeping when the mouse moves around *within* the Close button.) When the mouse leaves the close button—either because it moved to another part of the window or because it left the window entirely—we reset the flag.

When you run this program, it pretty much behaves as intended. But that's because we haven't tried anything interesting yet.

Merge in the changes from our sample drag/drop program, so now you have a program that both performs drag/drop and which has special Close button behavior.

Now things get interesting. Run the program and drag out of the client area (triggering the drag/drop behavior) and hover the mouse over the Close button.

Ow, my ears!

What happened here?

When the drag/drop loop is in progress, the mouse is captured to the drag/drop window. Mouse capture means that all mouse messages go to that window (for as long as a mouse button is held down). "I don't care what window you think the mouse is over; it's over me!" Another way of looking at this is that the capture window logically covers the entire screen (for the purpose of determining who gets the mouse message).

The drag/drop loop wants to know which window is under the drag cursor so it can figure out whose *IDropTarget* should receive the drag/drop notifications. This *WindowFromPoint* call triggers a `WM_NCHITTEST` message, which our program incorrectly interprets as a "the mouse is now in my window". (Since the mouse is captured, the mouse really isn't in your window; it's in the window that has capture because that window is stealing all the mouse input.) It then performs its "The mouse is in the Close button" activities (BEEP). But since the mouse *was never in the window to begin with*, the `TrackMouseEvent` call that requests "let me know when the mouse leaves my window" posts a `WM_NCMOUSELEAVE` message immediately. The window then cleans up its "mouse is in the Close button" behaviors, ready for the next cycle.

And the next cycle begins pretty much as soon as the previous cycle finished, because the mouse is still physically (but not logically) in the Close button.

Result: Infinite beep loop.

(The real-life situation that triggered this article was much more complicated than this, involving an animation rather than a beep, but the result was effectively the same: Under the right circumstances, just moving the mouse over the caption resulted in the animation becoming an epileptic-seizure-inducing flicker as the animation continuously started and stopped.)

As we saw some time ago, the `WM_MOUSEMOVE` message is the way to detect that the mouse has entered your window. (Though some people haven't figured this out and continue on their fruitless quest for the `WM_MOUSEENTER` message.)

In our case, the applicable message is `WM_NCMOUSEMOVE` rather than `WM_MOUSEMOVE`, since we are operating on the nonclient area. Therefore, the fix is to move the code that starts the animation from `WM_NCHITTEST` to `WM_NCMOUSEMOVE`.

```
// Delete the old OnNcHitTest function and replace it with this
void OnNcMouseMove(HWND hwnd, int x, int y, UINT codeHitTest)
{
 FORWARD_WM_NCMOUSEMOVE(hwnd, x, y, codeHitTest, DefWindowProc);
 if (codeHitTest == HTCLOSE) {
  EnterCloseButton(hwnd);
 } else {
  LeaveCloseButton(hwnd);
 }
 return ht;
}
// delete HANDLE_MSG(hwnd, WM_NCHITTEST, OnNcHitTest);
HANDLE_MSG(hwnd, WM_NCMOUSEMOVE, OnNcMouseMove);
```

Remember, if you want to do something when the mouse enters your window, wait until the mouse actually enters your window. The `WM_NCHITTEST` message doesn't mean that the mouse is in your window; it just means that somebody is asking, "If the mouse *were* in your window, what would it be doing?"

Raymond Chen

Follow