Implementing higher-order clicks

devblogs.microsoft.com/oldnewthing/20041018-00

October 18, 2004

Raymond Chen

Another question people ask is “How do I do triple-click or higher?” Once you see the algorithm for double-clicks, extending it to higher order clicks should be fairly natural. The first thing you probably should do is to remove the `CS_DBLCLKS` style from your class because you want to do multiple-click management manually.

Next, you can simply reimplement the same algorithm that the window manager uses, but take it to a higher order than just two. Let’s do that. Start with a clean scratch program and add the following:

```
int g_cClicks = 0;
RECT g_rcClick;
DWORD g_tmLastClick;


void OnLButtonDown(HWND hwnd, BOOL fDoubleClick,
                   int x, int y, UINT keyFlags)
{
  POINT pt = { x, y };
  DWORD tmClick = GetMessageTime();


  if (!PtInRect(&g_rcClick, pt) ||
      tmClick – g_tmLastClick > GetDoubleClickTime()) {
    g_cClicks = 0;
  }
  g_cClicks++;


  g_tmLastClick = tmClick;
  SetRect(&g_rcClick, x, y, x, y);
  InflateRect(&g_rcClick,
              GetSystemMetrics(SM_CXDOUBLECLK) / 2,
              GetSystemMetrics(SM_CYDOUBLECLK) / 2);


  TCHAR sz[20];
  wnsprintf(sz, 20, TEXT(“%d”), g_cClicks);
  SetWindowText(hwnd, sz);
}


void ResetClicks()
{
  g_cClicks = 0;
  SetWindowText(hwnd, TEXT(“Scratch”));
}


void OnActivate(HWND hwnd, UINT state, HWND, BOOL)
{
  ResetClicks();
}


void OnRButtonDown(HWND hwnd, BOOL fDoubleClick,
                   int x, int y, UINT keyFlags)
{
  ResetClicks();
}
```

```
    HANDLE_MSG(hwnd, WM_LBUTTONDOWN, OnLButtonDown);
    HANDLE_MSG(hwnd, WM_ACTIVATE, OnActivate);
```

[Boundary test for double-click time corrected 10:36am.]

(Our scratch program doesn't use the `CS_DBLCLKS` style, so we didn't need to remove it – it wasn't there to begin with.)

The basic idea here is simple: When a click occurs, we see if it is in the "double-click zone" and has occurred within the double-click time. If not, then we reset the consecutive click count.

(Note that the SM_CXDOUBLECLK and SM_CYDOUBLECLK values are the width of the entire rectangle, so we cut it in half when inflating so that the rectangle extends halfway in either direction. Yes, this means that a pixel is lost if the double-click width is odd, but Windows has been careful always to set the value to an even number.)

Next, we record the coordinates and time of the current click so the next click can compare against it.

Finally, we react to the click by putting the consecutive click number in the title bar.

There are some subtleties in this code. First, notice that setting `g_cClicks` to zero forces the next click to be treated as the first click in a series, for regardless of whether it matches the other criteria, all that will happen is that the click count increments to 1.

Next, notice that the way we test whether the clicks occurred within the double click time was done in a manner that is not sensitive to timer tick rollover. If we had written

```
    tmClick > g_tmLastClick + GetDoubleClickTime()) {
```

then we would fail to detect multiple clicks properly near the timer tick rollover. (Make sure you understand this.)

Third, notice that we reset the click count when the window gains or loses activation. That way, if the user clicks, then switches away, then switches back, and then clicks again, that is not treated as a double-click. We do the same if the user clicks the right mouse button in between. (You may notice that few programs bother with quite this much subtlety.)

Exercise: Suppose your program isn't interested in anything beyond triple-clicks. How would you change this program in a manner consistent with the way the window manager stops at double-clicks?

Raymond Chen

Follow