Autoscrolling on drag, part 1: Basic implementation

devblogs.microsoft.com/oldnewthing/20210125-00

January 25, 2021

Raymond Chen

Many windows perform autoscroll on drag, typically for the purpose of allowing you to drag-select a region of the window that is larger than the current client area. There are quite a few subtleties here, so I'm going to start with a basic implementation and then dig into it.

Start with the scratch program and first add this code for basic scroll bar support. For further discussion of this code, go back to the scroll bars series that started this blog.

```
int g_yOrigin;              /* Scrollbar position */
int g_cyContent = 10000;
int g_cxContent = 1000;
int g_cyLine;
int g_cyWindow;

void ScrollTo(HWND hwnd, int pos)
{
    pos = max(pos, 0);
    pos = min(pos, g_cyContent - g_cyWindow);

    ScrollWindowEx(hwnd, 0, g_yOrigin - pos,
        NULL, NULL, NULL, NULL,
        SW_ERASE | SW_INVALIDATE);

    g_yOrigin = pos;

    SCROLLINFO si;
    si.cbSize = sizeof(si);
    si.fMask = SIF_PAGE | SIF_POS | SIF_RANGE;
    si.nPage = g_cyWindow;
    si.nMin = 0;
    si.nMax = g_cyContent - 1;     /* endpoint is inclusive */
    si.nPos = g_yOrigin;
    SetScrollInfo(hwnd, SB_VERT, &si, TRUE);
}

void ScrollDelta(HWND hwnd, int dpos)
{
    ScrollTo(hwnd, g_yOrigin + dpos);
}

void OnVScroll(HWND hwnd, HWND hwndCtl, UINT code, int pos)
{
    switch (code) {
    case SB_LINEUP:         ScrollDelta(hwnd, -g_cyLine); break;
    case SB_LINEDOWN:       ScrollDelta(hwnd, +g_cyLine); break;
    case SB_PAGEUP:         ScrollDelta(hwnd, -g_cyWindow); break;
    case SB_PAGEDOWN:       ScrollDelta(hwnd, +g_cyWindow); break;
    case SB_THUMBPOSITION:  ScrollTo(hwnd, pos); break;
    case SB_THUMBTRACK:     ScrollTo(hwnd, pos); break;
    case SB_TOP:            ScrollTo(hwnd, 0); break;
    case SB_BOTTOM:         ScrollTo(hwnd, MAXLONG); break;
    }
}

void
OnSize(HWND hwnd, UINT state, int cx, int cy)
{
    g_cyWindow = cy;
    ScrollDelta(hwnd, 0);
}
```

```
BOOL
OnCreate(HWND hwnd, LPCREATESTRUCT lpcs)
{
    g_cyLine = GetSystemMetrics(SM_CYHSCROLL);
    return TRUE;
}

void
PaintContent(HWND hwnd, PAINTSTRUCT* pps)
{
    POINT org;
    OffsetWindowOrgEx(pps->hdc, 0, g_yOrigin, &org);
    MoveToEx(pps->hdc, 0, 0, nullptr);
    LineTo(pps->hdc, g_cxContent, g_cyContent);
    MoveToEx(pps->hdc, g_cxContent, 0, nullptr);
    LineTo(pps->hdc, 0, g_cyContent);
    SetWindowOrgEx(pps->hdc, org.x, org.y, nullptr);
}

    HANDLE_MSG(hwnd, WM_VSCROLL, OnVScroll);
```

This code draws a giant X of a fixed width and height. The X is very tall, and the scroll bar lets you see the full height. (I'm not going to bother with the width. Use your imagination.)

Okay, we knew how to do this already. Now to do the autoscrolling.

The idea behind autoscrolling is that if the user drags the mouse out of the window, then the window starts scrolling, continuing until the mouse returns inside the window. This lets the user perform a drag-selection of content larger than the window.

```
bool g_fDragging = false;
```

This global variable keeps track of whether we have captured the mouse for dragging and possibly autoscrolling.

```
#define IDT_AUTOSCROLL 1

void CancelAutoScroll(HWND hwnd)
{
    KillTimer(hwnd, IDT_AUTOSCROLL);
}
```

Right now, all we need to do to cancel autoscroll is to cancel the autoscroll timer. Future versions of this program will eventually put more stuff here, but we'll just start with this.

```
void OnCancelMode(HWND hwnd)
{
    if (g_fDragging) {
        g_fDragging = false;
        ReleaseCapture();
        CancelAutoScroll(hwnd);
    }
}

    HANDLE_MSG(hwnd, WM_CANCELMODE, OnCancelMode);
```

If we are asked to cancel whatever we're doing, then we stop dragging by releasing the capture, and then we also cancel any autoscroll.

```
void OnLButtonDown(HWND hwnd, BOOL fDoubleClick, int x, int y, UINT keyFlags)
{
    if (DragDetect(hwnd, POINT{ x, y })) {
        SetCapture(hwnd);
        g_fDragging = true;
    }
}

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

When the user presses the left mouse button, we check to see if it is the start of a drag operation. If so, then we capture the mouse (allowing us to receive mouse activity even if the mouse moves outside the window), and remember that we are dragging.

```
void OnLButtonUp(HWND hwnd, int x, int y, UINT keyFlags)
{
    OnCancelMode(hwnd);
}

    HANDLE_MSG(hwnd, WM_LBUTTONUP, OnLButtonUp);
```

When the left button is released, we exit drag mode.

```
BOOL TryAutoScroll(HWND hwnd, POINT pt)
{
    if (pt.y < 0) {
        ScrollDelta(hwnd, -g_cyLine);
        return TRUE;
    } else if (pt.y > g_cyWindow) {
        ScrollDelta(hwnd, +g_cyLine);
        return TRUE;
    }
    return FALSE;
}
```

The `TryAutoScroll` function actually performs the autoscroll operation if applicable. If the mouse is above the top of the window, then scroll up. If it is below the bottom, then scroll down. And we let the caller know whether we performed any scrolling.

The magic happens when the mouse moves.

```
void HandleDragMouseMove(HWND hwnd, POINT pt)
{
    if (TryAutoScroll(hwnd, pt)) {
        SetTimer(hwnd, IDT_AUTOSCROLL,
                 GetDoubleClickTime() / 5, OnAutoScroll);
    } else {
        KillTimer(hwnd, IDT_AUTOSCROLL);
    }
}
```

When the mouse moves when we are dragging, we ask the `TryAutoScroll` function to perform any necessary autoscroll. If it did, then we start the autoscroll timer; otherwise we cancel it.

The autoscroll rate is traditionally based on the user's double-click time, because that is a general indicator of how fast the user's reaction time is.

All that's left is hooking things up.

```
void OnMouseMove(HWND hwnd, int x, int y, UINT keyFlags)
{
    if (g_fDragging) {
        HandleDragMouseMove(hwnd, { x, y });
    }
}

    HANDLE_MSG(hwnd, WM_MOUSEMOVE, OnMouseMove);
```

When the mouse moves during a drag, we ask `HandleDragMouseMove` to figure out what to do.

```
void CALLBACK OnAutoScroll(HWND hwnd, UINT message, UINT_PTR id, DWORD time)
{
    POINT pt;
    if (GetCursorPos(&pt) && ScreenToClient(hwnd, &pt)) {
        HandleDragMouseMove(hwnd, pt);
    }
}
```

The autoscroll timer drives the autoscrolling. We get the current cursor position, convert it to client coordinates, and then let `HandleDragMouseMove` decide what to do. That might exit autoscroll mode, which is fine. Autoscroll will start up again the next time the mouse leaves the window.

Okay, so that's the basics of autoscroll. Next time, we'll look at one of the unintended consequences of this design.

Raymond Chen

Follow