Scrollbars, part 4: Adding a proportional scrollbar

devblogs.microsoft.com/oldnewthing/20030731-00

July 31, 2003

Raymond Chen

To obtain a proportional scrollbar, you need to tell Windows the minimum and maximum values covered by the scrollbar, the current scrollbar position, and the size of the scrollbar thumb (called the "page size"). One annoyance of the way scrollbars are set up is that the maximum value is attainable. This differs from the way GDI manages dimensions, where the range is exclusive of the endpoint. As a result, there will be occasional "-1″s sprinkled through the code to compensate for the fact that scrollbars include rather than exclude their endpoints.

To do this, we need a few more variables.

```
int g_yOrigin;              /* Scrollbar position */
int g_cLinesPerPage;        /* Number of lines per page */
```

The name of the variable *g_yOrigin* will become apparent later.

Next comes the helper function that is at the center of the scrollbar action. Given the desired position of the scrollbar, it sanitizes the value, scrolls the window contents as necessary, and sets the scrollbar parameters to match the new state of the window.

```
void ScrollTo(HWND hwnd, int pos)
{
    /*
     *  Keep the value in the range 0 .. (g_cItems - g_cLinesPerPage).
     */
    pos = max(pos, 0);
    pos = min(pos, g_cItems - g_cLinesPerPage);
    /*
     *  Scroll the window contents accordingly.
     */
    ScrollWindowEx(hwnd, 0, (g_yOrigin - pos) * g_cyLine,
                   NULL, NULL, NULL, NULL,
                   SW_ERASE | SW_INVALIDATE);
    /*
     *  Now that the window has scrolled, a new item is at the top.
     */
    g_yOrigin = pos;
    /*
     *  And make the scrollbar look like what we think it is.
     */
    SCROLLINFO si;
    si.cbSize = sizeof(si);
    si.fMask = SIF_PAGE | SIF_POS | SIF_RANGE;
    si.nPage = g_cLinesPerPage;
    si.nMin = 0;
    si.nMax = g_cItems - 1;     /* endpoint is inclusive */
    si.nPos = g_yOrigin;
    SetScrollInfo(hwnd, SB_VERT, &si, TRUE);
}
```

Sometimes, we merely want to make a relative change to the scrollbar position.

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

When the window changes size, we need to recompute the number of items that fit on one page. This in turn may require that the scrollbar thumb position be adjusted, so we also perform a dummy scroll, which triggers all the sanity-check computations.

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

The *WM_VSCROLL* handler is pretty much self-explanatory. When scrolling by lines or pages, we make a relative change in the appropriate direction and magnitude. When the user drags the thumb, we move directly to the specified location. And when the user drags to the top or bottom of the scrollbar, we peg at one or the other extremum.

```
void OnVscroll(HWND hwnd, HWND hwndCtl, UINT code, int pos)
{
    switch (code) {
    case SB_LINEUP:         ScrollDelta(hwnd, -1); break;
    case SB_LINEDOWN:       ScrollDelta(hwnd, +1); break;
    case SB_PAGEUP:         ScrollDelta(hwnd, -g_cLinesPerPage); break;
    case SB_PAGEDOWN:       ScrollDelta(hwnd, +g_cLinesPerPage); 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;
    }
}
```

And, of course, we need to hook up our message handler into the message loop.

```
    /* Add to WndProc */
    HANDLE_MSG(hwnd, WM_VSCROLL, OnVscroll);
```

Finally, we need to make our paint handler aware of the scrollbar. Fortunately, through the magic of GDI transforms, we can accomplish this without having to change the *PaintSimpleContent* function at all.

```
void PaintContent(HWND hwnd, PAINTSTRUCT *pps)
{
    POINT ptOrgPrev;
    OffsetRect(&pps->rcPaint, 0, g_yOrigin * g_cyLine);
    GetWindowOrgEx(pps->hdc, &ptOrgPrev);
    SetWindowOrgEx(pps->hdc, ptOrgPrev.x, ptOrgPrev.y + g_yOrigin * g_cyLine, NULL);
    PaintSimpleContent(hwnd, pps);
    SetWindowOrgEx(pps->hdc, ptOrgPrev.x, ptOrgPrev.y, NULL);
}
```

By changing the window origin, the *PaintSimpleContent* function continues to believe that there is no scrollbar at all. It blithely draws item zero at (0, 0), but through the magic of GDI transforms, the pixels actually appear at the new origin location.

Now you see why the variable is named *g_yOrigin*.

Exercise: There is a latent bug in *OnVscroll*. Explain what it is and fix it.

Raymond Chen

Follow