Creating a window that can be resized in only one direction

devblogs.microsoft.com/oldnewthing/20150504-00

May 4, 2015

Raymond Chen

Today's Little Program shows a window that can be resized in only one direction, let's say vertically but not horizontally.

Start with the scratch program and make these changes:

```
UINT OnNcHitTest(HWND hwnd, int x, int y)
{
 UINT ht = FORWARD_WM_NCHITTEST(hwnd, x, y, DefWindowProc);
 switch (ht) {
 case HTBOTTOMLEFT:  ht = HTBOTTOM; break;
 case HTBOTTOMRIGHT: ht = HTBOTTOM; break;
 case HTTOPLEFT:     ht = HTTOP;    break;
 case HTTOPRIGHT:    ht = HTTOP;    break;
 case HTLEFT:        ht = HTBORDER; break;
 case HTRIGHT:       ht = HTBORDER; break;
 }
 return ht;
}

HANDLE_MSG(hwnd, WM_NCHITTEST, OnNcHitTest);
```

We accomplish this by removing horizontal resize behavior from the left and right edges and corners. For the corners, we remove the horizontal resizing, but leave the vertical resizing. For the edges, we remove resizing entirely by reporting that the left and right edges should act like an inert border.

Wait, we're not done yet. This handles resizing by grabbing the edges with the mouse, but it doesn't stop the user from hitting `Alt`+`Space`, followed by `S` (for Size), and then hitting the left or right arrow keys.

For that, we need to handle `WM_GETMINMAXINFO`.

```
void OnGetMinMaxInfo(HWND hwnd, LPMINMAXINFO lpmmi)
{
 RECT rc = { 0, 0, 500, 0 };
 AdjustWindowRectEx(&rc, GetWindowStyle(hwnd), FALSE,
                    GetWindowExStyle(hwnd));

 // Adjust the width
 lpmmi->ptMaxSize.x =
 lpmmi->ptMinTrackSize.x =
 lpmmi->ptMaxTrackSize.x = rc.right - rc.left;
}

HANDLE_MSG(hwnd, WM_GETMINMAXINFO, OnGetMinMaxInfo);
```

This works out great, except for the case of being maximized onto a secondary monitor, because we run into the mixed case of being small than the monitor in the horizontal direction, but larger than the monitor in the vertical direction.

```
void OnGetMinMaxInfo(HWND hwnd, LPMINMAXINFO lpmmi)
{
 RECT rc = { 0, 0, 500, 0 };
 AdjustWindowRectEx(&rc, GetWindowStyle(hwnd), FALSE,
                    GetWindowExStyle(hwnd));

 // Adjust the width
 lpmmi->ptMaxSize.x =
 lpmmi->ptMinTrackSize.x =
 lpmmi->ptMaxTrackSize.x = rc.right - rc.left;

 // Adjust the height
 MONITORINFO mi = { sizeof(mi) };
 GetMonitorInfo(MonitorFromWindow(hwnd,
                    MONITOR_DEFAULTTOPRIMARY), &mi);
 lpmmi->ptMaxSize.y = mi.rcWork.bottom - mi.rcWork.top
                    - lpmmi->ptMaxPosition.y + rc.bottom;
}
```

The math here is a little tricky. We want the window height to be the height of the work area of the window monitor, plus some extra goop in order to let the borders hang over the edge.

The first two terms are easy to explain: `mi.rcWork.bottom - mi.rcWork.top` is the height of the work area.

Next, we want to add the height consumed by the borders that hang off the top of the monitor. Fortunately, the window manager told us exactly how much the window is going to hang off the top of the monitor: It's in `lpmmi->ptMaxPosition.y`, but as a negative value since it is a coordinate that is off the top of the screen. We therefore have to negate it before adding it in.

Finally, we add the borders that hang off the bottom of the work area.

Yes, handling this mixed case (where the window is partly constrained and partly unconstrained) is annoying. Sorry.

Raymond Chen

Follow