What's the point of DeferWindowPos?

devblogs.microsoft.com/oldnewthing/20050706-26

July 6, 2005

Raymond Chen

The purpose of the `DeferWindowPos` function is to move multiple child windows at one go. This reduces somewhat the amount of repainting that goes on when windows move around.

Take that DC brush sample from a few months ago and make the following changes:

```
HWND g_hwndChildren[2];


BOOL
OnCreate(HWND hwnd, LPCREATESTRUCT lpcs)
{
 const static COLORREF s_rgclr[2] =
    { RGB(255,0,0), RGB(0,255,0) };
 for (int i = 0; i < 2; i++) {
  g_hwndChildren[i] = CreateWindow(TEXT("static"), NULL,
        WS_VISIBLE | WS_CHILD, 0, 0, 0, 0,
        hwnd, (HMENU)IntToPtr(s_rgclr[i]), g_hinst, 0);
  if (!g_hwndChildren[i]) return FALSE;
 }
 return TRUE;
}
```

Notice that I'm using the control ID to hold the desired color. We retrieve it when choosing our background color.

```
HBRUSH OnCtlColor(HWND hwnd, HDC hdc, HWND hwndChild, int type)
{
  Sleep(500);
  SetDCBrushColor(hdc, (COLORREF)GetDlgCtrlID(hwndChild));
  return GetStockBrush(DC_BRUSH);
}


    HANDLE_MSG(hwnd, WM_CTLCOLORSTATIC, OnCtlColor);
```

I threw in a half-second sleep. This will make the painting a little easier to see.

```
void
OnSize(HWND hwnd, UINT state, int cx, int cy)
{
  int cxHalf = cx/2;
  SetWindowPos(g_hwndChildren[0],
               NULL, 0, 0, cxHalf, cy,
               SWP_NOZORDER | SWP_NOOWNERZORDER | SWP_NOACTIVATE);
  SetWindowPos(g_hwndChildren[1],
               NULL, cxHalf, 0, cx-cxHalf, cy,
               SWP_NOZORDER | SWP_NOOWNERZORDER | SWP_NOACTIVATE);
}
```

We place the two child windows side by side in our client area. For our first pass, we'll use the `SetWindowPos` function to position the windows.

Compile and run this program, and once it's up, click the maximize box. Observe carefully which parts of the green rectangle get repainted.

Now let's change our positioning code to use the `DeferWindowPos` function. The usage pattern for the deferred window positioning functions is as follows:

```
HDWP hdwp = BeginDeferWindowPos(n);
if (hdwp) hdwp = DeferWindowPos(hdwp, …); // 1 [fixed 7/7]
if (hdwp) hdwp = DeferWindowPos(hdwp, …); // 2
if (hdwp) hdwp = DeferWindowPos(hdwp, …); // 3
…
if (hdwp) hdwp = DeferWindowPos(hdwp, …); // n
if (hdwp) EndDeferWindowPos(hdwp);
```

There are some key points here.

- The value you pass to the `BeginDeferWindowPos` function is the number of windows you intend to move. It's okay if you get this value wrong, but getting it right will reduce the number of internal reallocations.
- The return value from `DeferWindowPos` is stored back into the `hdwp` because the return value is not necessarily the same as the value originally passed in. If the deferral bookkeeping needs to perform a reallocation, the `DeferWindowPos` function returns a handle to the new defer information; the old defer information is no longer valid. What's more, if the deferral fails, the old defer information is **destroyed**. This is different from the `realloc` function which leaves the original object unchanged if the reallocation fails. The pattern `p = realloc(p, …)` is a memory leak, but the pattern `hdwp = DeferWindowPos(hdwp, …)` is not.

That second point is important. Many people get it wrong.

Okay, now that you're all probably scared of this function, let's change our repositioning code to take advantage of deferred window positioning. It's really not that hard at all. (Save these changes to a new file, though. We'll want to run the old and new versions side by side.)

```
void
OnSize(HWND hwnd, UINT state, int cx, int cy)
{
  HDWP hdwp = BeginDeferWindowPos(2);
  int cxHalf = cx/2;
  if (hdwp) hdwp = DeferWindowPos(hdwp, g_hwndChildren[0],
               NULL, 0, 0, cxHalf, cy,
               SWP_NOZORDER | SWP_NOOWNERZORDER | SWP_NOACTIVATE);
  if (hdwp) hdwp = DeferWindowPos(hdwp, g_hwndChildren[1],
               NULL, cxHalf, 0, cx-cxHalf, cy,
               SWP_NOZORDER | SWP_NOOWNERZORDER | SWP_NOACTIVATE);
  if (hdwp) EndDeferWindowPos(hdwp);
}
```

Compile and run this program, and again, once it's up, maximize the window and observe which regions repaint. Observe that there is slightly less repainting in the new version compared to the old version.

Raymond Chen

Follow