Separating the metadata from the DIB pixels: Precalculating the BITMAPINFO

devblogs.microsoft.com/oldnewthing/20090715-00

July 15, 2009

Raymond Chen

Last time, we saw that you can use the `SetDIBitsToDevice` function to draw a DIB with an alternate color table without having to modify the `HBITMAP`. In that version of the function, we selected the `HBITMAP` into a device context in preparation for drawing from it, but in fact that step isn't necessary for drawing. It was merely necessary to get the original color table so we could build our grayscale color table. If you don't care what the original colors are, then you can skip that step. And even if you care what the old colors are, and if you assume that the colors don't change, then you only need to ask once.

To demonstrate, that all the work of building the `BITMAPINFO` structure could have been done ahead of time, let's use this alternate version of our program:

```
HBITMAP g_hbm;
struct BITMAPINFO256 {
 BITMAPINFOHEADER bmiHeader;
   RGBQUAD bmiColors[256];
} g_bmiGray;
void *g_pvBits;
BOOL
OnCreate(HWND hwnd, LPCREATESTRUCT lpcs)
{
 // change path as appropriate
 g_hbm = (HBITMAP)LoadImage(g_hinst,
                      TEXT("C:\\Windows\\Gone Fishing.bmp"),
                      IMAGE_BITMAP, 0, 0,
                      LR_CREATEDIBSECTION | LR_LOADFROMFILE);
 if (g_hbm) {
  BITMAP bm;
  if (GetObject(g_hbm, sizeof(bm), &bm) == sizeof(bm) &&
                bm.bmBits != NULL &&
                bm.bmPlanes * bm.bmBitsPixel <= 8) {
   ZeroMemory(&g_bmiGray, sizeof(g_bmiGray));
   HDC hdc = CreateCompatibleDC(NULL);
   if (hdc) {
    HBITMAP hbmPrev = SelectBitmap(hdc, g_hbm);
    UINT cColors = GetDIBColorTable(hdc, 0, 256, g_bmiGray.bmiColors);
    for (UINT iColor = 0; iColor < cColors; iColor++) {
     BYTE b = (BYTE)((30 * g_bmiGray.bmiColors[iColor].rgbRed +
                      59 * g_bmiGray.bmiColors[iColor].rgbGreen +
                      11 * g_bmiGray.bmiColors[iColor].rgbBlue) / 100);
     g_bmiGray.bmiColors[iColor].rgbRed   = b;
     g_bmiGray.bmiColors[iColor].rgbGreen = b;
     g_bmiGray.bmiColors[iColor].rgbBlue  = b;
    }
    g_bmiGray.bmiHeader.biSize        = sizeof(g_bmiGray.bmiHeader);
    g_bmiGray.bmiHeader.biWidth       = bm.bmWidth;
    g_bmiGray.bmiHeader.biHeight      = bm.bmHeight;
    g_bmiGray.bmiHeader.biPlanes      = bm.bmPlanes;
    g_bmiGray.bmiHeader.biBitCount    = bm.bmBitsPixel;
    g_bmiGray.bmiHeader.biCompression = BI_RGB;
    g_bmiGray.bmiHeader.biClrUsed     = cColors;
    g_pvBits                          = bm.bmBits;
    DeleteDC(hdc);
   }
 }
 return TRUE;
}
void
PaintContent(HWND hwnd, PAINTSTRUCT *pps)
{
 if (g_pvBits) {
    SetDIBitsToDevice(pps->hdc, 0, 0,
                  g_bmiGray.bmiHeader.biWidth,
                  g_bmiGray.bmiHeader.biHeight, 0, 0,
                  0, g_bmiGray.bmiHeader.biHeight,
                  g_pvBits,
                  (BITMAPINFO*)&g_bmiGray, DIB_RGB_COLORS);
 }
}
```

I moved the blue code from `PaintContent` to `OnCreate` to demonstrate that pretty much all of the work we used to do in `PaintContent` could have been done ahead of time. The only other thing we had to do was save the pointer to the bits so we could pass them to `SetDIBitsToDevice`. (Of course, that pointer becomes invalid once the controlling `HBITMAP` is destroyed, so be careful! In practice, you probably would be better off calling `GetObject` immediately before drawing to protect against the case that somebody deleted the bitmap out from under you.)

Next time, we'll look at another operation we can perform when we have a `BITMAPINFO` and a collection of pixels.

(Note that there are issues with this technique which will be taken up on Friday.)